Outils d'aide à la décision pour des problèmes d'ordonnancement dynamiques

Abdallah Elkhyari

UNIVERSITÉ DE NANTES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATÉRIAUX

Année : 2003 N^0 B.U. : .....

Thèse de Doctorat de l'Université de Nantes

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

ABDALLAH ELKHYARI

à l'École Nationale Supérieure
des Techniques Industrielles et des Mines de Nantes

Outils d'aide à la décision pour des problèmes d'ordonnancement dynamiques

SOUTENUE LE 27 NOVEMBRE 2003 devant la commission d'Examen

Composition du jury

Président	:	**Frédéric**	**Benhamou**	Professeur, Université de Nantes
Rapporteurs	:	**Gérard**	**Verfaillie**	Directeur de recherches, LAAS-CNRS
	:	**Aziz**	**Moukrim**	Maître de conférences, UTC

Directeur de thèse	:	**Patrice Boizumault**
Laboratoire	:	GREYC, CNRS UMR 6072, Université de Caen
Co-encadrant	:	**Christelle Guéret**
Laboratoire	:	Département Automatique-Productique, École des Mines de Nantes
Co-encadrant	:	**Narendra Jussien**
Laboratoire	:	Département Informatique, École des Mines de Nantes

N^o ED 366 - 121

Remerciements

Ce travail a été réalisé au sein de l'équipe *Contraintes* du département Informatique, et de l'équipe *Systèmes Logistiques et de Production* du département Automatique Productique de l'École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes.

Je tiens tout d'abord à remercier Monsieur Patrice Boizumault, mon directeur de thèse, Professeur à l'Université de Caen, pour m'avoir encadré tout au long de cette thèse. J'ai beaucoup apprécié la liberté qu'il m'a laissée dans mes recherches et la confiance qu'il m'a accordée. Qu'il soit aussi remercié pour sa gentillesse, sa disponibilité et ses encouragements.

Je tiens à remercier très sincèrement mes deux co-encadrants : Madame Christelle Guéret, Maître de conférences au département Automatique Productique de l'École des Mines de Nantes, et Monsieur Narendra Jussien, Maître de conférences au département Informatique de l'École des Mines de Nantes. Le suivi de mes travaux, leurs conseils avisés, les échanges riches et de haut niveau, sont autant d'attentions qui m'ont assuré des conditions idéales pour mener à bien ces années de thèse. Qu'ils trouvent ici l'expression de ma plus sincère gratitude.

J'adresse mes sincères remerciements à Monsieur Gérard Verfaillie, Directeur de recherches au *Laboratoire d'Analyse et d'Architecture des Systèmes LAAS-CNRS* de Toulouse, ainsi qu'à Monsieur Aziz Moukrim, Maître de conférences à l'Université de Technologie de Compiègne, pour l'intérêt qu'ils ont porté à mon travail en acceptant d'être rapporteurs de cette thèse. J'ai particulièrement apprécié les discussions enrichissantes avec chacun d'eux, tant du point de vue scientifique que sur le plan humain. Leurs remarques constructives sur mon travail m'ont été et continuent de m'être très profitables. Merci également à Monsieur Frédéric Benhamou, Professeur à l'Université de Nantes, qui m'a fait l'honneur de présider le jury de ma thèse.

Je voudrais également remercier tous les membres des deux équipes *Contraintes* et *Systèmes Logistiques et de Production*, ainsi qu'à tous les doctorants du département Informatique, pour leur gentillesse, leur amitié et leur soutien.

Je ne remercierai jamais assez *ma femme* qui m'a supporté pendant toutes ces années de thèse, et qui m'a beaucoup aidé et encouragé.

Bien sûr, je voudrais remercier *ma famille*, en particulier *mes parents*, à qui je dois beaucoup.

Table des matières

Chapitre 1

Introduction

1.1 Contexte : un environnement dynamique de résolution de contraintes

Les problèmes de satisfaction de contraintes (CSP) constituent un domaine de recherche en pleine expansion. Ils fournissent un cadre permettant de traiter un grand nombre de problèmes issus de l'intelligence artificielle et de la recherche opérationnelle. Ils sont de plus en plus utilisés dans le secteur industriel car ils offrent une souplesse pour la modélisation et permettent de résoudre des problèmes complexes. Ainsi, plusieurs problèmes d'optimisation dans le domaine industriel ont été résolus efficacement : ordonnancement, diagnostic de panne, aide à la décision, ...

De façon générale, un problème de satisfaction de contraintes consiste à trouver, parmi les valeurs possibles d'un ensemble de variables, celles qui satisfont un ensemble de contraintes. Le principe de la résolution repose sur une exploration de l'espace des solutions en utilisant des algorithmes visant à réduire et à parcourir intelligemment l'espace de recherche.

De nombreux problèmes réels sont sujet à des perturbations. Les systèmes d'aide à la décision, qui permettent de faire face à ces perturbations, sont de plus en plus demandés par les industriels. Nous en distinguons deux types : des systèmes qui ne raisonnent que sur des données internes, prennent leurs propres décisions puis informent l'utilisateur, et des systèmes qui permettent un dialogue entre l'utilisateur et le solveur. L'utilisateur peut alors commander le système en lui fournissant de nouvelles données et le solveur informe l'utilisateur en retour.

Lorsqu'un problème, modélisé comme un problème de satisfaction de contraintes, subit des perturbations (au cours de la résolution ou pendant l'exécution de la solution), l'ensemble des contraintes le définissant peut changer : de nouvelles contraintes sont imposées ou d'anciennes contraintes sont retirées. La solution obtenue risque de devenir invalide. Il est donc nécessaire de construire une nouvelle solution respectant les nouvelles contraintes du problème. Pour remédier à cette difficulté des techniques permettant de résoudre les problèmes de satisfaction de contraintes dynamiques (Dynamic Constraint Satisfaction Problems ou DCSP) ont été développées.

Un DCSP peut être défini comme une séquence de CSP dont chaque élément diffère du précédent par l'ajout et/ou le retrait de certaines contraintes. De nombreuses techniques ont été proposées pour la résolution des DCSP. Leur objectif commun est d'éviter de recommencer la recherche à zéro après chaque modification du problème. La démarche suivie consiste à mémoriser et à réutiliser des résultats obtenus lors des recherches précédentes.

L'ajout successif de contraintes dans un problème de satisfaction de contraintes peut conduire à un système de contraintes sans solution (ou contradictoire). Nous parlons alors de CSP sur-contraint. Dans ce cas, l'objectif est de trouver une solution qui ne viole qu'un ensemble minimal (suivant un critère donné) de contraintes.

Récemment, la notion d'*explication* a été introduite dans le but de résoudre les DCSP et les CSP sur-contraints. Une explication est un ensemble de contraintes lié à un événement donné (retrait d'une valeur dans le domaine d'une variable, modification d'une borne d'une variable, un échec dans la résolution ...) tel que l'événement associé est sûr de se produire dès que les contraintes de l'explication sont actives dans le système de contraintes. Les explications permettent de défaire les effets passés d'une contrainte dans un système de contraintes, ce qui facilite le retrait incrémental des contraintes. Les explications sont aussi utilisables pour résoudre les problèmes sur-contraints car ils permettent de déterminer un sous-ensemble de contraintes responsable de cet échec. L'utilisateur peut donc retirer incrémentalement des contraintes de ce sous-ensemble jusqu'à obtenir une solution.

La notion d'explication a montré son intérêt dans plusieurs applications : résolution de problèmes dynamiques et sur-contraints, amélioration d'algorithmes de recherche, ...

1.2 Problématique : ordonnancement dynamique

Un problème d'ordonnancement consiste à organiser dans le temps un ensemble d'activités, de façon à satisfaire un ensemble de contraintes et à optimiser une ou plusieurs fonctions objectifs. La plupart des travaux réalisés en ordonnancement considèrent que les caractéristiques du problème (activités, ressources, contraintes ...) sont totalement connues à l'avance et ne varient pas. En réalité, ces problèmes sont très souvent soumis aux aléas (une nouvelle commande à prendre en compte, une panne de machine, ...). Lorsqu'une telle situation se produit, une nouvelle solution doit être calculée en un temps raisonnable. De plus, devant la compétitivité que connaît le secteur industriel, cette nouvelle solution doit satisfaire au mieux un certain critère (par exemple : minimiser le nombre d'activités dont la date de début a changé, minimiser le nombre de séquences dont l'ordre a changé). Pour cela, des mesures de performance ont été introduites. Nous en distinguons deux types : *robustesse* et *stabilité*.

La prise en compte du caractère dynamique de ces problèmes est un champ de recherche nouveau dans lequel peu de résultats existent à ce jour.

1.3 Notre thèse : nouveaux outils pour résoudre le RCPSP dynamique et quelques extensions

L'objectif de cette thèse est de développer un système interactif d'aide à la décision permettant de résoudre des problèmes d'ordonnancement dynamiques.

Dans cette thèse, nous considérons un problème d'ordonnancement très général : le RCPSP. Ce problème consiste à ordonnancer un ensemble d'activités tout en respectant des contraintes temporelles (contraintes de précédence reliant les activités) et des contraintes de ressources (respect des capacités des ressources utilisées par les activités). Nous considérons ici comme objectif la minimisation de la durée totale du projet. Ce problème a de nombreuses applications (ordonnancement d'atelier, gestion de projets, emploi du temps, ...) car il recouvre un grand nombre de problèmes d'ordonnancement.

Nous nous plaçons de plus dans un environnement dynamique dans lequel des événements inattendus peuvent survenir à tout moment. Ces événements peuvent être par exemple une panne de machine ou l'arrivée d'une commande urgente s'il s'agit d'un problème de production, ou bien l'indisponibilité d'un professeur dans le cas d'un problème d'emploi du temps. Nous supposons donc que des activités, des ressources et des contraintes temporelles peuvent être ajoutées, retirées ou modifiées. Ce problème est habituellement appelé **ordonnancement de projet à contraintes de ressource dynamique** *(dynamic resource-constrained project scheduling problem)*.

Les techniques utilisées pour la résolution des CSP ont montré leur utilité pour représenter et résoudre certain problèmes d'ordonnancement *NP-difficiles*. Parmi les techniques utilisées, nous citons : la propagation de contraintes, les stratégies de résolution portant sur le choix des variables à instancier, le choix des valeurs pour une variable, les techniques de retour arrière, ... Aussi, nous proposons dans cette thèse d'apporter des contributions pour la résolution des problèmes d'ordonnancement dynamiques, en particulier les RCPSP, en nous basant sur des travaux réalisés pour la résolution des CSPs dynamiques, en particulier ceux utilisant comme outils les *explications*.

1.4 Contributions et organisation de la thèse

Ce document est organisé de la façon suivante :

La **première partie** dresse un état de l'art des travaux portant d'une part sur les problèmes d'ordonnancement auxquels nous nous intéressons : le RCPSP et les problèmes d'ordonnancement dynamiques; d'autre part sur des outils utilisés pour résoudre les CSP dynamiques et sur-contraints. Cette première partie est donc composée de quatre chapitres : le premier chapitre décrit le problème RCPSP statique ainsi que les nombreuses approches et techniques qui ont été développées pour le résoudre. Le second chapitre présente les différents types de méthodes développées pour prendre en compte l'aspect dynamique des problèmes d'ordonnancement. Dans le chapitre 3, nous introduisons les CSP et présentons différentes approches développées pour résoudre les CSP dynamiques. Enfin dans le chapitre 4 nous présentons en détail une technique utilisée pour résoudre les CSP dynamiques et que nous allons utiliser dans le cadre des problèmes d'ordonnancement dynamiques : les explications. Nous terminons cette partie par un bilan de ces différents états de l'art.

La **deuxième partie** présente nos contributions pour la résolution du RCPSP dynamique. Dans cette partie nous présentons notre environnement de résolution du RCPSP dans le cadre dynamique. Le premier chapitre décrit les techniques utilisées pour la conception de notre système : un schéma de séparation inspiré d'un schéma de séparation de la littérature, la notion de distance qui nous a permis de modéliser de façon simple les contraintes temporelles du problème (ici contraintes de précédence) ainsi que toutes les contraintes de décision définissant notre schéma de séparation et leurs négations, et enfin l'algorithme de résolution utilisé qui est une généralisation de l'algorithme *mac-dbt*. Le second chapitre traite les contraintes temporelles : précédence, chevauchement et disjonction. Nous présentons leur mécanisme de propagation et le calcul des explications associées. Ces contraintes sont utilisées dans la définition du problème, au cours de la recherche (ajout et/ou retrait des contraintes) et pour la prise en compte des aléas. Dans le troisième chapitre nous présentons deux contraintes globales permettant de satisfaire les contraintes de ressources : une basée sur les deux notions *parties obligatoires* et *histogramme* et l'autre sur la notion d'*intervalles de tâches*. Nous dé-

crivons de plus le système permettant de maintenir les structures de données pour chacune de ces contraintes, leur mécanisme de propagation, ainsi que le calcul des explications associées. Le dernier chapitre est consacré à la validation de notre système pour la résolution du RCPSP dynamique. Il contient les résultats de plusieurs séries de tests sur des problèmes de la littérature dans le cadre statique puis dynamique.

Dans la **troisième partie**, nous présentons nos contributions pour la résolution des extensions du RCPSP dynamique. Cette partie présente des extensions des contraintes temporelles et de ressources développées dans la partie précédente. Ces contraintes sont intégrées dans notre système. Dans le premier chapitre, nous présentons une généralisation des contraintes temporelles (précédence, chevauchement et disjonction) en introduisant des délais constants ou bornés par deux valeurs entre deux activités. Nous présentons de plus le mécanisme de propagation de ces contraintes et le calcul des explications associées. Dans le second chapitre nous proposons une extension des deux contraintes de ressources développée pour le RCPSP. Ces contraintes permettent de maintenir la consommation en ressource dans le cas où les disponibilités des ressources ne sont plus constantes mais variables. L'intégration de ces deux contraintes dans notre système permettent de résoudre plusieurs extensions du RCPSP. Dans le dernier chapitre nous présentons les résultats obtenus pour la résolution du RCPSP avec des contraintes temporelles généralisées et le GRCPSP dans le cadre dynamique.

Première partie

État de l'art

Introduction

Cette partie a pour but de dresser un état de l'art des méthodes développées pour résoudre d'une part le RCPSP statique et les problèmes d'ordonnancement dynamiques, d'autre part les CSP, CSP dynamiques et CSP sur-contraints.

Un RCPSP est un problème d'ordonnancement très général dans lequel un ensemble d'activités doivent être ordonnancées de façon à minimiser un certain critère, tout en respectant des contraintes de précédence entre ces activités et des contraintes de ressources.

Un CSP est défini par un ensemble de variables, un ensemble de domaines associés à ses variables et un ensemble de contraintes portant sur ses variables. Le but est de trouver une affectation des valeurs des domaines aux variables de façon à satisfaire toutes les contraintes.

Cette partie est organisée de la façon suivante :

• Le premier chapitre a pour objectif de présenter le RCPSP, et de résumer les principales méthodes de résolution du cas statique. En particulier, nous décrivons des méthodes pour le calcul de bornes inférieures, des méthodes heuristiques, des méthodes exactes, ainsi que des règles d'élimination. Enfin, nous présentons quelques extensions du RCPSP.

• Dans le deuxième chapitre, nous présentons les différentes approches existant dans la littérature pour la résolution des problèmes d'ordonnanceemnt en présence de perturbations. Nous donnons un état de l'art de ces approches en nous basant sur la classification de *Davenport et Beck* [2000].

• Le troisième chapitre présente le formalisme CSP et les différentes techniques utilisées dans la littérature pour sa résolution. Nous présentons aussi les problématiques particulières liées à l'optimisation, la résolution des problèmes dynamiques et la résolution des problèmes sur-contraints.

• Dans le dernier chapitre, nous détaillons une des techniques, présentée dans le chapitre 4, basée sur la notion d'*explication*, et décrivons le système PaLM, l'environnement de programmation de notre système.

Chapitre 2

RCPSP statique : présentation et état de l'art

Sommaire

Introduction

Après avoir défini le problème d'ordonnancement de projet à contraintes de ressource (ou RCPSP), nous décrivons dans ce chapitre, tout d'abord des bornes inférieures, puis des méthodes approchées et exactes pour la résolution du RCPSP, ainsi que différentes règles de déduction. Enfin, quelques extensions du RCPSP sont présentées.

2.1 Présentation du RCPSP

Le RCPSP est un problème classique en ordonnancement qui a fait l'objet de nombreux travaux. Ceci est dû au fait qu'il est très présent dans les problématiques industrielles. Citons par exemple : les systèmes de production comme les problèmes d'atelier, la gestion de projet, le problème de découpe, etc.

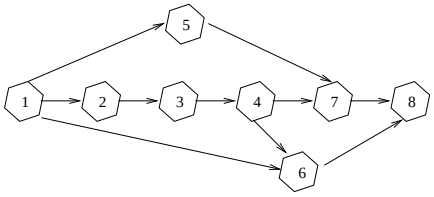
2.1.1 Définition

Le RCPSP est formulé par la donnée d'un ensemble d'activités $A = \{1, 2, .., n\}$ et d'un ensemble de ressources renouvelables[1] $R = \{1, .., r\}$. Chaque ressource k est disponible en une quantité constante R_k. Chaque activité i de A a une durée opératoire p_i et nécessite pour sa réalisation une quantité constante a_{ik} de chaque ressource k.

Dans un RCPSP, nous distinguons des **contraintes temporelles** qui sont définies par des relations de précédence, et des **contraintes de ressources** qui exigent qu'à chaque instant et pour chaque ressource, la demande totale en ressource ne dépasse pas la quantité de ressource disponible.

Dans notre étude, la préemption n'est pas autorisée et la fonction objectif considérée est la minimisation de la durée totale du projet. Ce problème est noté $PS \mid prec \mid C_{max}$ (*Brucker et al.* [1999]), ou encore $m, 1 \mid cpm \mid C_{max}$ (*Herroelen et al.* [1998]).

La figure 2.1 représente un exemple de RCPSP constitué de huit activités utilisant deux ressources de capacité $R_1 = 3$ et $R_2 = 4$ respectivement. Le graphe représente les précédences entre les activités et le tableau fournit les durées opératoires et les quantités de ressource associées à chacune des huit activités.



i	1	2	3	4	5	6	7	8
p_i	0	2	4	2	8	4	4	0
a_{i1}	0	2	1	2	1	3	1	0
a_{i2}	0	3	2	1	2	4	2	0

FIG. 2.1 – *Un exemple de* RCPSP.

La figure 2.2 représente une solution réalisable du RCPSP définie dans la figure 2.1. Dans cette figure, l'axe des abscisses représente le temps et l'axe des ordonnées représente le niveau de consommation de ressource. Pour chacune des deux ressources, une activité est représentée par un rectangle de longueur égale à sa durée opératoire et de largeur égale à son besoin de ressource. Les parties grisées représentent les surfaces libres.

1. Une ressource est dite renouvelable si elle est de nouveau disponible après son utilisation par une activité.

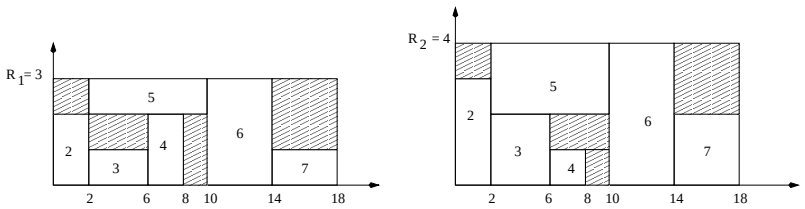


FIG. 2.2 – *Une solution réalisable de l'exemple (Figure 2.1).*

Notons que plusieurs problèmes d'optimisation peuvent être modélisés comme un RCPSP ou des cas particuliers de ce problème. Par exemple : les problèmes d'ordonnancement à une machine, les problèmes d'ordonnancement à machines identiques parallèles, les problèmes d'ateliers [Brucker, 1999] et les problèmes d'emploi du temps [Brucker et Kampmeyer, 2003] [Brucker, 1999].

2.1.2 Complexité

Le RCPSP généralise plusieurs problèmes d'ordonnancement classiques *NP-difficiles*. Il est donc lui-aussi *NP-difficile* (au sens fort) [Blazewicz *et al.*, 1983].

2.1.3 Notations

Le tableau 2.1 liste les notations qui seront utilisées dans la suite de ce mémoire.

2.2 Bornes inférieures

Disposer d'une borne inférieure est essentiel lorsque l'on veut résoudre des problèmes *NP-difficiles*. Cela permet de donner une mesure de la performance des heuristiques utilisées. Cela permet aussi de limiter l'arbre de recherche en excluant les nœuds qui ne peuvent pas conduire à une solution meilleure que la solution trouvée. Concernant le RCPSP, nous pouvons distinguer deux types de stratégies pour le calcul d'une borne inférieure : les méthodes constructives et les méthodes destructives.

2.2.1 Méthodes constructives

Ces méthodes calculent une borne inférieure en résolvant un problème relaxé du problème initial. En voici quelques exemples :

• **Bornes inférieures basées sur le chemin critique :** Une des premières bornes qui a été utilisée pour le RCPSP est obtenue en calculant la longueur d'un chemin critique CC dans le graphe de précédence. Cette borne a été améliorée par *Stinson et al.* [1978] et *Demeulemeester et Herroelen* [1992] en considérant une ou plusieurs activités supplémentaires à ordonnancer en parallèle avec les activités du chemin critique.

Brucker et al. [1996] calculent une borne inférieure en résolvant un problème de *Job-Shop* à deux machines obtenu en considérant deux chemins disjoints du graphe de précédence, l'un de ces chemins étant un chemin critique.

Symbole	*Description*
$A = \{1, .., n\}$	est l'ensemble des activités. Les activités 1 et n sont fictives et représentent respectivement le début et la fin du projet.
$R = \{1, .., r\}$	est l'ensemble des ressources renouvelables.
R_k	est une quantité constante représentant la disponibilité de la ressource k.
p_i	est la durée opératoire de l'activité i.
a_{ik}	est la quantité de ressource k nécessaire pour réaliser l'activité i.
$w_{ik} = a_{ik} \times p_i$	est le travail (ou quantité d'énergie) nécessaire pour réaliser l'activité i sur la ressource k.
r_i	est la date de début au plus tôt de l'activité i.
f_i	est la date de début au plus tard de l'activité i.
d_i	est la date échue de l'activité i.
t_i	est la date de début de l'activité i.
C_i	est la date de fin de l'activité i.
S_i	est l'ensemble des successeurs de l'activité i.
$i \rightarrow j$	signifie que l'activité j ne peut commencer avant la fin de l'activité i (i et j sont en *conjonction*).
$i \parallel j$	signifie que les deux activités i et j se chevauchent pendant au moins une unité de temps (i et j sont dites en *parallèle*).
$i \leftrightarrow j$	signifie que les deux activités i et j ne peuvent pas s'exécuter en parallèle (i et j sont en *disjonction*).

TAB. 2.1 – *Liste des notations.*

• **Borne inférieure basée sur la notion d'énergie :** Pour une ressource k donnée, le rapport $\lceil \frac{\sum_{i=1}^{n} w_{ik}}{R_k} \rceil$ est une borne inférieure du RCPSP.

• **Borne inférieure basée sur une relaxation en problèmes d'ordonnancement à machines parallèles :** La méthode proposée par *Carlier et Latapie* [1991] consiste à générer, pour chaque ressource k, des ensembles d'activités tels qu'au plus m activités de chacun de ces ensembles peuvent être exécutées en même temps. À chacun de ces ensembles est associé un problème à m machines parallèles (problème *NP-difficile*) pour lequel les auteurs calculent une borne inférieure en utilisant l'*algorithme préemptif de Jackson* [Carlier et Pinson, 1994]. Une borne inférieure du RCPSP est la valeur maximale des bornes inférieures associées aux problèmes générés.

• **Borne inférieure basée sur une formulation par programme linéaire :** *Mingozzi et al.* [1998] proposent une formulation du RCPSP en programme linéaire en $0-1$. Les variables correspondent à tous les sous-ensembles d'activités pouvant être exécutées simultanément sans violer les contraintes de précédence ou de ressources. Différentes relaxations sont utilisées, l'une d'entre elles aboutit à un problème de *stable de poids maximal* (*weigthed independent set problem*), problème *NP-difficile* pour lequel les auteurs calculent une borne inférieure en utilisant un algorithme de liste.

2.2.2 Méthodes destructives

Le principe de cette méthode est de prouver qu'il n'existe pas d'ordonnancement de durée totale inférieure ou égale à une borne inférieure BI (qui peut être calculée par une méthode constructive). La borne BI n'est alors plus valide et peut être augmentée. L'augmentation de la borne BI se fait soit par incrémentation, soit par dichotomie.

- l'incrémentation augmente la valeur de BI à $BI + \Delta$ (où $\Delta \geqslant 1$).

- la recherche dichotomique nécessite une borne supérieure UB. S'il est possible de prouver qu'il n'existe pas d'ordonnancement réalisable de durée totale inférieure ou égale à $UB' = \lfloor \frac{UB+BI}{2} \rfloor$, alors BI peut être remplacée par $UB' + 1$. Sinon UB est remplacée par UB'.

Ce processus est réitéré jusqu'à ce qu'on ne puisse plus prouver qu'il n'existe pas d'ordonnancement de durée totale inférieure ou égale à BI. Pour cela, des conditions nécessaires d'existence d'une solution sont utilisées. En voici quelques exemples :

• **La relaxation élastique :** Pour appliquer cette technique au RCPSP, il faut dans un premier temps le relaxer en un problème d'ordonnancement cumulatif (*Cumulative Scheduling Problem* ou CuSP) en relaxant toutes les contraintes de ressources sauf une et en traduisant les contraintes de précédence par des dates de disponibilité et des dates échues pour les activités.

Cette technique a été appliquée par *Baptiste* [1998] et *Néron* [1999]. Le principe de cette relaxation consiste à considérer que chaque activité i du CuSP obtenu exige pour sa réalisation des quantités de ressources non pas constantes mais variables dans le temps. Si $c_i(t)$, $t \in [r_i, d_i]$, désigne la quantité de ressource k nécessaire à l'activité i à l'instant t alors $\sum_{t \in [r_i, d_i]} c_i(t) = w_{ik}$.

Les auteurs proposent plusieurs relaxations élastiques du CuSP en imposant différentes contraintes sur la fonction $c_i(t)$. En se basant sur ces différentes relaxations, ils proposent des conditions d'existence d'une solution réalisable du CuSP initial.

• **Borne inférieure basée sur les parties obligatoires :** Étant donnée une borne inférieure BI, les dates de début au plus tôt r_i et de début au plus tard f_i de chaque activité i sont calculées. La partie obligatoire *PO(i)* [Lahrichi, 1982], associée à une activité i, est la longueur de l'intervalle durant lequel une partie de l'activité i est exécutée quelle que soit sa date de démarrage dans $[r_i, f_i]$ (*Fig.* 2.3).

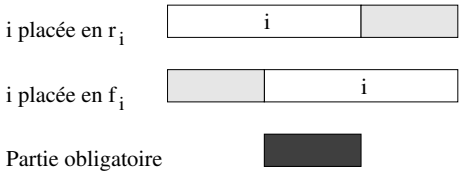


FIG. 2.3 – *La partie obligatoire associée à une activité.*

Dans [Klein et Scholl, 1999], les auteurs ont utilisé la notion de partie obligatoire (*core-times*) pour réduire la borne inférieure BI. La procédure de réduction consiste à ordonnancer tous les *PO(i)*. Si un conflit de ressource existe alors la valeur de la borne BI est augmentée. Sinon, pour chaque activité, une fenêtre temporelle durant laquelle cette activité peut être entièrement ordonnancée est recherchée. Lorsqu'il n'existe qu'un seul intervalle possible pour une activité i, sa fenêtre temporelle est alors réduite à cet intervalle et les fenêtres temporelles de ses successeurs et prédécesseurs sont ajustées. Ceci peut avoir pour effet la modification des parties obligatoires. Lorsqu'il n'existe pas d'intervalle, la borne BI peut être augmentée.

La procédure est réitérée après chaque augmentation de la borne inférieure BI ou chaque amélioration des parties obligatoires.

• **Borne inférieure combinant la programmation par contraintes et la programmation linéaire :** *Brucker et Knust* [2000] ont présenté un algorithme combinant deux approches. La première est une technique de programmation par contraintes. Elle utilise des règles de déductions basées sur une notion de distance définie comme une évaluation par défaut de la différence entre les dates de début de deux activités. La seconde reprend la formulation en programme linéaire introduite par *Mingozzi et al.* (*cf.* section 2.2.1 page 9). Les auteurs résolvent une relaxation de ce programme linéaire en utilisant une méthode de *génération de colonnes*.

2.3 Méthodes heuristiques

Les méthodes de résolution approchées (ou méthodes heuristiques) sont utilisées pour la résolution de problèmes *NP-difficiles* et de grandes tailles. Elles sont aussi utilisées pour calculer des bornes supérieures dans les recherches arborescentes, permettant ainsi d'éliminer des nœuds ne pouvant conduire à une solution meilleure que la solution trouvée.

Les méthodes heuristiques pour le RCPSP sont très variées. Voici les principales.

2.3.1 Heuristiques basées sur des règles de priorité

Une heuristique basée sur une règle de priorité consiste à ordonnancer les activités les unes après les autres suivant un ordre de priorité donné. Les composantes d'une telle heuristique sont : la *règle de priorité* et le *schéma de génération d'ordonnancements (schedule generation scheme)*.

2.3.1.1 Quelques règles de priorité

De nombreuses règles de priorité peuvent être utilisées. Nous pouvons distinguer par exemple celles liées au graphe de précédence, aux dates de début ou aux besoins de ressources des activités.

Nous avons aussi plusieurs types de règles, par exemple, des règles locales ne tenant compte que de l'activité considérée, des règles globales tenant compte de plusieurs activitées liées à l'activité considérée, des règles statiques construisant la liste de priorité une fois pour toute au début de l'algorithme et des règles dynamiques pour lesquelles la liste de priorité évolue au cours des itérations. Pour le RCPSP, plus de *70* règles de priorité ont été présentées. Le tableau 2.2 liste les règles de priorité les plus utilisées.

règles de priorité statiques basées sur le graphe	
SPT	$\min\{p_i\}$
shortest processing time	[Alvarez-Valdés et Tamrit, 1996]
GRPW	$\max\{p_i + \sum_{j\in S_i} p_j\}$
greatest rank positional weight	[Alvarez-Valdés et Tamrit, 1996]
règles de priorité statiques basées sur le chemin critique	
LST	$\min\{f_i\}$
latest starting time	[Kolisch, 1996a]
LFT	$\min\{f_i + p_i\}$
latest finishing time	[Davies, 1973]
MSL	$\min\{f_i - r_i\}$
minimum slack time	[Davies, 1973]
règles de priorité basées sur les quantités de ressources	
GRD	$\max\{p_i \times \sum_{k=1}^{r} r_{ik}\}$
greatest resource demand	[Ulusoy et Özdamar, 1989]
WRUP	$\max\{\omega\times \mid S_i \mid +(1-\omega)\times \sum_{k=1}^{r} \frac{r_{ik}}{R_k}\}$
weighted resource utilization and precedence	[Ulusoy et Özdamar, 1989]
règles de priorités statiques composées	
ACROS	$\max\{\sum_{j\in\Pi_{ih}} \sum_{k=1}^{r} \frac{r_{ik}}{R_k}\}$
—	[Bedworth, 1973]
ACTRES	$\max\{\sum_{j\in\Pi_{ih}} \sum_{k=1}^{r} \frac{r_{ik}\times p_i}{R_k}\}$
—	[Elsayed et Nasr, 1986]

TAB. 2.2 – *Quelques règles de priorité utilisées pour le RCPSP*

2.3.1.2 Le schéma de génération d'ordonnancements

Le schéma de génération d'ordonnancements décrit l'algorithme utilisé pour ordonnancer les activités une fois que la règle de priorité est fixée. Deux variantes sont à distinguer : le *schéma de génération d'ordonnancements en série (serial schedule generation scheme)* et le *schéma de génération d'ordonnancements en parallèle (parallel schedule generation scheme).*

Le *schéma de génération d'ordonnancements en série* consiste à déterminer, à chaque itération, l'ensemble E_g des activités éligibles, *i.e.* l'ensemble des activités dont tous les prédécesseurs sont déjà ordonnancés. L'activité de E_g de plus grande priorité est sélectionnée et ordonnancée le plus tôt possible de façon à respecter les contraintes de précédence et de ressource. Le processus est répété jusqu'à ce que toutes les activitées soient ordonnancées.

Le *schéma de génération d'ordonnancements en parallèle* ordonnance plusieurs activités à chaque itération. Il consiste à déterminer l'ensemble $E(t)$ des activités pouvant démarrer à l'instant t correspondant à la plus petite date de fin des activités ordonnancées à l'itération précédente. Ces activités sont alors sélectionnées les unes après les autres suivant l'ordre de la règle de priorité choisie et placées à l'instant t si elles ne violent pas de contraintes de ressource. Si les quantités de ressources disponibles ne sont pas suffisantes pour exécuter une activité, celle-ci est retirée de l'ensemble. Elle sera ordonnancée dans une itération ultérieure.

Kolisch et Hartmann [1998] ont montré que le *schéma de génération d'ordonnancements en série* génère des ordonnancements actifs, *i.e.* aucune activité ne peut être exécutée plus tôt sans retarder d'autres activités, et que le *schéma de génération d'ordonnancements partiel* génère des ordonnancements sans délai (*no-delay*), dans lesquels aucune ressource ne reste libre si elle peut executer une activité. Notons que dans le cas où la fonction objectif est de minimiser le *makespan*, l'ensemble des ordonnancements actifs contient au moins un ordonnancement optimal, ce qui n'est pas le cas pour l'ensemble des ordonnancements sans délai.

2.3.1.3 Quelques méthodes proposées dans le littérature

Les premières heuristiques présentées dans la littérature sont une combinaison d'un *schéma de génération d'ordonnancement* et d'une seule *règle de priorité* (*Alvarez-Valdés et Tamrit* [1996], *Cooper* [1976], *Davis et Patterson* [1975], *Kolisch* [1996a, 1996b], *Lawrence* [1985], *Valls et al.* [1992]). Ce type d'heuristiques construit à chaque itération un seul ordonnancement partiel.

Un autre type d'heuristique consiste à utiliser un *schéma de génération d'ordonnancement* avec plusieurs *règles de priorité* ([Thomas et Salhi, 1997] [Ulusoy et Özdamar, 1989] [Boctor, 1990]) et à conserver la meilleure des solutions obtenues. Une approche générant un nombre illimité d'ordonnancements partiels a été proposée par *Ulusoy et Özdamar* [1989]. Cette approche consiste à utiliser des combinaisons convexes des valeurs attribuées aux activités par les règles de priorité. Si $v_m(i)$ désigne la priorité associée à l'activité i suivant la règle de priorité m alors à l'activité i est associée la valeur $v(i) = \sum_{m=1}^{M} w_m \times v_m(i)$ avec $w_m \geqslant 0 \ \forall m \in M$ et $\sum_{m=1}^{M} w_m = 1$. Les auteurs ont utilisé cette approche pour deux règles de priorité en faisant varier les coefficients w_m dans le but de générer à chaque itération *10* ordonnancements différents.

2.3.2 Heuristiques basées sur les arcs disjonctifs

Ces heuristiques se basent sur les ensembles interdits. Un ensemble interdit est un ensemble d'activités qui ne peuvent pas être exécutées simultanément. Le principe de cette méthode consiste à étendre l'ensemble des arcs conjonctifs en orientant les arcs disjonctifs de façon à éliminer les ensembles interdits minimaux.

Alvarez-Valdes et Tamarit [1996] ont proposé quatre stratégies différentes pour éliminer les ensembles interdits minimaux. Leur meilleure stratégie commence par les ensembles interdits

minimaux de cardinalité inférieure. Un ensemble est arbitrairement choisi et est éliminé en orientant les arcs disjonctifs de façon à minimiser la date de fin au plus tôt de l'activité fictive n.

2.3.3 Méta-heuristiques

Un grand nombre de méta-heuristiques ont été développées pour le RCPSP. Ainsi, *Baar et al.* [1998] ont développé deux algorithmes de recherche *Tabou*. Dans ces algorithmes, les listes *Taboues* utilisées sont dynamiques (la longueur de la liste *Taboue* peut varier. La liste est augmentée lorsqu'un cycle est détecté, est diminuée lorsqu'un blocage est rencontré) et la solution initiale est calculée par une heuristique basée sur des règles de priorité. Dans le premier algorithme, une solution est représentée par une liste d'activités qui peut être obtenue à partir de l'ordonnancement en concordance avec le *schéma de génération d'ordonnancements en série*. Le voisinage est défini par une liste des activités qui peuvent être relaxées en décalant l'activité critique vers une nouvelle position. La seconde approche est basée sur le schéma de séparation introduit par *Brucker et al.* [1998, 1999] (*cf.* section 2.4.4 page 14). Les voisins sont obtenus en forçant des paires d'activités à être en parallèle ou en imposant des contraintes de disjonction entre des paires d'activités en parallèle.

Lee et Kim [1996] ont proposé un voisinage utilisé dans trois méta-heuristiques : *un recuit simulé*, *une recherche Tabou* et *un algorithme génétique*. Pour ces trois stratégies, une méthode basée sur une règle de priorité est utilisée pour générer un ordonnancement. Une solution du voisinage est obtenue en changeant les priorités entre les activités.

2.4 Méthodes exactes

Les méthodes exactes sont utilisées pour produire des solutions optimales. Elles peuvent aussi servir à mesurer les performances des méthodes heuristiques en comparant le coût de la solution approchée avec celui de la solution optimale. Dans la littérature, il existe trois types de méthodes exactes : la programmation mathématique, la programmation dynamique et la procédure par séparation et évaluation (*Branch and Bound*).

La méthode la plus utilisée pour la résolution exacte du RCPSP est la procédure par séparation et évaluation. Cette procédure nécessite une borne inférieure et/ou une borne supérieure et un schéma de séparation. Nous avons vu dans les sections précédentes comment calculer des bornes inférieures (*cf.* section 2.2 page 7) et des bornes supérieures (*cf.* section 2.3 page 10). Nous présentons, dans cette section, quelques schémas de séparation existants pour le RCPSP.

2.4.1 Schéma de séparation chronologique

Un schéma de séparation chronologique consiste à ajouter une à une des activités dans l'ordonnancement partiel. Ainsi un nœud de l'arborescence correspond à un ensemble d'activités pour lesquelles les dates de début ont été fixées.

Baptiste et Le Pape [1997] ont utilisé un schéma de séparation chronologique pour le RCPSP. En chaque nœud P, l'ensemble des activités éligibles $EL(P)$ est déterminé, *i.e.* $EL(P)$ est l'ensemble des activités dont tous les prédécesseurs sont ordonnancés. Une activité de $EL(P)$ est choisie et placée au plus petit instant possible de façon à respecter les contraintes de précédence et les contraintes de ressources. De plus, cet instant doit être supérieur à l'instant

de placement de la dernière activité. Lors d'un retour arrière (*backtrack*), une contrainte imposant à au moins une des autres activités éligibles de commencer avant la date de début de l'activité choisie est ajoutée.

2.4.2 Décalage d'un ensemble d'activités

Demeulemeester et Herroelen [1992] ont proposé un schéma de séparation utilisant des sous-ensembles d'activités minimaux pour résoudre les conflits de ressource. Chaque activité est exécutée le plus tôt possible en respectant les contraintes de précédence et de ressource. À chaque nœud de l'arborescence correspond un couple *(OP,t)* où *OP* est un ordonnancement partiel et t est la plus petite date de fin des activités ordonnancées au niveau précédent. À chaque instant t, l'ensemble *E(OP,t)* des activités éligibles n'appartenant pas à l'ordonnancement partiel *OP* et dont tous les prédécesseurs sont ordonnancés est déterminé. Si aucun conflit n'a lieu en ordonnançant toutes les activités de *E(OP,t)* à l'instant t, un fils unique avec l'ordonnancement partiel obtenu est créé. S'il y a un conflit, toutes les alternatives minimales sont déterminées. Un sous-ensemble d'activités est une alternative si les conflits de ressource sont résolus en différant les instants d'exécution de ces activités. Une alternative est dite minimale si elle ne contient aucun sous-ensemble alternatif. À chaque alternative minimale est associée un nouveau fils dont l'ordonnancement partiel est obtenu en ajoutant des relations de précédence.

2.4.3 Réduction des fenêtres temporelles

Carlier et Latapie [1991] ont proposé un schéma de séparation utilisant les fenêtres temporelles des activités. À chaque nœud de l'arborescence correspond un ensemble d'activités ayant chacune une fenêtre temporelle $[r_i, d_i]$. En chaque nœud, une activité i est selectionnée, sa marge $M_i = d_i - p_i - r_i$ est calculée, et deux fils f_1 et f_2 sont créés :

- dans f_1 la date de disponibilité de l'activité i est modifiée en posant : $r_i = r_i + \frac{M_i}{2}$.
- dans f_2 la date échue de l'activité i devient : $d_i = d_i - \frac{M_i}{2}$.

La règle de *Jackson* [Carlier et Pinson, 1988] est ensuite appliquée pour déterminer un ordonnancement compatible avec les dates de disponibilité des activités. Ces dates sont mises à jour lorsqu'une nouvelle solution est trouvée. Une feuille est atteinte lorsque toutes les activités ont une marge nulle.

L'efficacité de ce schéma de séparation dépend du critère utilisé pour choisir l'activité dont la fenêtre temporelle est modifiée. Les auteurs ont utilisé le critère suivant : en chaque nœud P, une borne inférieure *BI(P)* associée à ce nœud et les bornes inférieures $BI(f_1)$ et $BI(f_2)$ associées à ses deux fils f_1 et f_2 sont calculées. L'activité qui maximise la somme des deux expressions $BI(f_1) - BI(P)$ et $BI(f_2) - BI(P)$ est alors sélectionnée.

2.4.4 Conjonctions, parallélismes et disjonctions

Dans [Brucker *et al.*, 1998, Brucker, 1999], les auteurs décomposent l'ensemble des paires d'activités en quatre sous-ensembles disjoints :

- C est constitué des paires d'activités *(i,j)* qui sont en conjonction, *i.e.* $i \rightarrow j$. C est initialisé par les conjonctions définies dans le graphe de précédence.

- D contient les paires d'activités *(i,j)* qui sont en disjonction, *i.e.* $i \leftrightarrow j$. D est initialisé par les paires d'activités qui ne pouvent pas s'exécuter en parallèle à cause des contraintes de ressources.

- N est l'ensemble des paires d'activités *(i,j)* qui sont en parallèle, *i.e.* $i \parallel j$. N est initialisé avec les paires d'activités qui satisfont la relation de parallélisme suivante : $si\ p_i + p_j > max\{d_i, d_j\} - min\{r_i, r_j\}\ alors\ i \parallel j$.

- F est constitué des paires d'activités *(i,j)*, dites flexibles, pour lesquelles aucune décision n'a été prise, *i.e.* $i \sim j$. F est initialisé avec les paires d'activités n'appartenant ni à C, ni à D, ni à N.

Les nœuds de l'arborescence sont caractérisés par des quadruplets *(C, D, N, F)*. Chaque nœud génére deux fils en choisissant une paire d'activités *(i, j)* de F. Ces fils sont définis par les quadruplets :

- $(C, D \cup \{(i,j)\}, N, F \setminus \{(i,j)\})$ qui impose aux activités i et j d'être en disjonction.

- $(C, D, N \cup \{(i,j)\}, F \setminus \{(i,j)\})$ qui impose aux deux activités i et j d'être en parallèle.

Une feuille de l'arbre est atteinte lorsque l'ensemble F devient vide. À ce moment l'ensemble des paires d'activités en disjonction D est généralement non vide. À chaque feuille est associé un ensemble *S(C, D, N)* d'ordonnancements, non nécessairement réalisables, qui maintiennent les relations imposées par les ensembles *C, D* et *N*. Notons par S_f*(C, D, N)* l'ensemble des ordonnancements réalisables de *S(C, D, N)*, *i.e.* qui satisfont en plus les contraintes de ressources.

Pour prouver qu'il existe une solution réalisable dans une feuille, *i.e.* S_f*(C, D, N)* est non vide, les auteurs déterminent l'ensemble des ordonnancements qui dérivent de *S(C, D, N)* en orientant toutes les disjonctions de telle sorte que la relation de conjonction soit transitive. Cet ensemble est dit *transitive orientation*. Pour une *transitive orientation S(C', ∅, N)* et pour chaque activité i, la valeur r_i du plus long chemin allant de *0* à i est calculée dans le graphe conjonctif associé. L'ordonnancement S_{ES}*(C')*, qui peut ne pas être réalisable, dans lequel chaque activité i débute à l'instant r_i est construit. D'après un résultat dû à *Möhring* [1985], si l'ordonnancement S_{ES}*(C')* est réalisable alors il domine tous les ordonnancements de S_f*(C, D, N)*. Sinon l'ensemble S_f*(C, D, N)* est vide.

2.5 Règles de déduction

Introduites tout d'abord par *Carlier et Pinson* [1989] pour les problèmes de *Job-Shop*, les règles de déduction sont des règles utilisées dans le but d'orienter les disjonctions et d'ajuster les fenêtres temporelles des activités. Voici quelques règles proposées dans la littérature pour le RCPSP.

2.5.1 Les arbitrages sur ensemble ascendants/descendants (*edge-finding*)

Ces règles reposent sur le fait que dans toute solution réalisable et pour toute clique K (ensemble d'activités deux à deux en disjonction), la différence entre la date de fin au plus

tard maximale et la date de début au plus tôt minimale des activités de K doit être supérieure à la somme des durées opératoires des activités de K.

Définition 1 (*edge-finding*)
Soit K une clique contenant au moins deux activités et l une activité de K.

- *l est dite* ***input*** *de la clique K ssi toutes les autres activités de la clique sont ordonnancées après l.*
- *l est dite* ***output*** *de la clique K ssi toutes les autres activités de la clique sont ordonnancées avant l.*
- *l est dite* ***middle*** *de la clique K ssi l n'est ni* input, *ni* output *dans K.*

Pinson [1988] a montré les règles suivantes :

$$\max_{i \in K \setminus \{l\}} \{d_i\} - r_l < \sum_{i \in K} p_i \Rightarrow l \ n'est \ pas \ input \ dans \ K$$

$$d_l - \min_{i \in K \setminus \{l\}} \{r_i\} < \sum_{i \in K} p_i \Rightarrow l \ n'est \ pas \ output \ dans \ K$$

$$\max_{i \in K \setminus \{l\}} \{d_i\} - \min_{i \in K \setminus \{l\}} \{r_i\} < \sum_{i \in K} p_i \Rightarrow l \ n'est \ pas \ middle \ dans \ K$$

De ces règles sont déduites les implications suivantes :

- l n'est ni *input*, ni *output* dans la clique $K \Rightarrow l$ est *middle* dans K.
- l n'est ni *input*, ni *middle* dans la clique $K \Rightarrow l$ est *output* dans K.
- l n'est ni *output*, ni *middle* dans la clique $K \Rightarrow l$ est *input* dans K.

Toutes ces règles permettent les ajustements suivants :

- l est *input* (ou *output*) dans la clique $K \Rightarrow$ des contraintes temporelles entre l et les autres activités de la clique K sont ajoutées.
- l n'est pas *input* $\Rightarrow$ la date de début au plus tôt de l est ajustée à la date de fin au plus tôt minimale de l'activité de $K \setminus \{l\}$ qui peut être ordonnancée en premier.
- l n'est pas *output* $\Rightarrow$ la date de fin au plus tard de l est ajustée à la date de début au plus tard maximale de l'activité de K qui peut être ordonnancée en dernier.
- l est *middle* $\Rightarrow$ l n'est ni *input*, ni *output*. Dans ce cas la date de début au plus tôt et la date de fin au plus tard de l peuvent être ajustées.

La technique d'*edge-finding* est très puissante, mais elle est très coûteuse. Elle n'est applicable que sur des problèmes de taille raisonnable.

Brucker [1997] a utilisé ces règles de déduction pour le RCPSP. Sa démarche consiste à appliquer ces règles sur les cliques du graphe représentant les positions relatives (précédence, parallélisme et disjonction) entre les activités du problème.

2.5.2 Les intervalles de tâches (*task-intervals*)

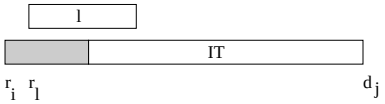
Introduite par *Caseau et Laburthe* [1994] pour les problèmes purement disjonctifs, la notion d'*intervalle de tâches* a pour but de réduire le nombre de sous-ensembles à tester afin d'appliquer la technique d'*edge-finding*. À chaque couple d'activités *(i, j)*, utilisant la même machine (ou ressource), est associé un ensemble d'activités défini comme suit :

Définition 2 (Intervalles de tâches)
Soit i et j deux activités distinctes (ou non) satisfaisant :
(1) i et j utilisent la même ressource.
(2) $r_i \leq r_j$ et $d_i \leq d_j$.
L'intervalle de tâches $IT = [i, j]$ est constitué de l'ensemble des activités l telles que :
(1) l utilise la même ressource que i.
(2) $r_i \leq r_l$ et $d_l \leq d_j$.

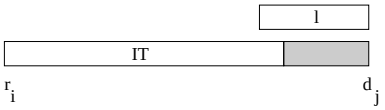
En appliquant la technique d'*edge-finding* aux intervalles de tâches, deux ensembles de règles ont été développées :

• **Des règles entre les activités d'un même intervalle de tâches :** Pour chaque activité l de l'intervalle de tâches IT, ces règles déterminent si l'activité l précède ou succède toutes les activités de $IT \backslash \{l\}$. Pour cela, les auteurs utilisent les règles de déduction présentées dans la section 2.5.1.

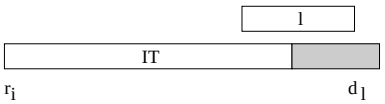
• **Des règles entre les activités d'un intervalle de tâches et une activité extérieure :** Ces règles déterminent si une activité l extérieure à IT doit être exécutée après, avant ou à l'intérieur de IT. Ceci est possible grâce aux trois tests suivants :



$$d_j < r_l + p_l + \sum_{l' \in IT} p_{l'} \Rightarrow l \text{ n'est pas avant } IT$$



$$d_j < r_i + p_l + \sum_{l' \in IT} p_{l'} \Rightarrow l \text{ n'est pas à l'intérieur de } IT$$



$$d_l < r_i + p_l + \sum_{l' \in IT} p_{l'} \Rightarrow l \text{ n'est pas après } IT$$

FIG. 2.4 – *Règles déduites des techniques d'edge-finding*

De ces propositions des ajustements peuvent être déduits. Par exemple :

- l n'est pas avant IT et l n'est pas à l'interieur de $IT \Rightarrow l$ est après IT

  et nous avons l'ajustement : $r_l := max\{r_l, \min_{l' \in IT \setminus \{l\}}(r_{l'}) + \sum_{l' \in IT \setminus \{l\}} p_{l'}\}$

- l ne peut pas être avant IT mais peut être à l'intérieur de $IT \Rightarrow l$ est placée après au moins une activité de IT

  et nous avons l'ajustement : $r_l := max\{r_l, \min_{l' \in IT}(r_{l'} + p_{l'})\}$

Dans [Caseau et Laburthe, 1996b], les auteurs ont étendu le principe d'*intervalle de tâches* aux problèmes cumulatifs en associant à chaque intervalle de tâches IT, lié à une ressource k, une énergie *energy(IT,k)* égale à la somme des énergies w_{lk} ($w_{lk} = a_{lk} \times p_l$) de toutes les activités l appartenant à l'intervalle de tâches IT. Les règles suivantes ont été établies :

• **Règle d'intégrité (*integrity rule*)** : Cette règle détecte une contradiction lorsque l'énergie associée au $IT = [i, j]$ dépasse la quantité d'énergie disponible, à savoir $(d_j - r_i) \times R_k$.

• **Règle de précédence (*precedence rule*)** : Cette règle consiste à appliquer le test suivant :

$$i \rightarrow j \ et \ r_j < r_i + p_i \Rightarrow r_j := r_i + p_i$$

• **Règles de jet (*throw rule*)** : Son principe consiste à caler à gauche (ou à droite) une activité l qui intersecte IT et à comparer la quantité d'énergie *overlap(IT,k,l)* intersectant IT à la quantité d'énergie disponible $slack(IT,k) = (d_j - r_i) \times R_k - energy(IT,k)$ (*Fig.* 9.12 et *Fig.* 9.13).

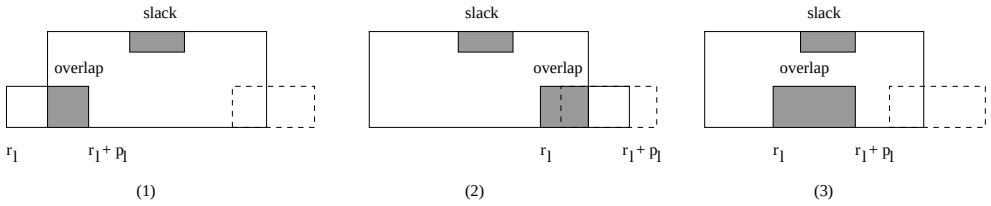


FIG. 2.5 – *Règles de jet : activité calée à gauche*

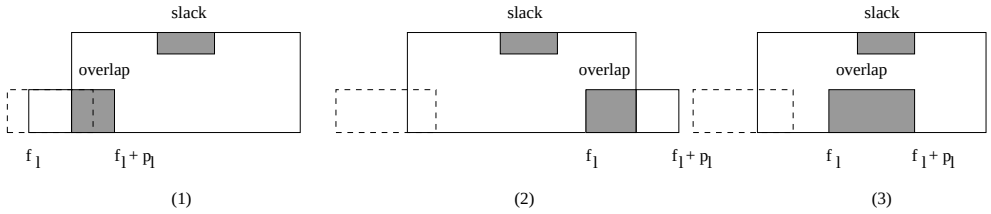


FIG. 2.6 – *Règles de jet : activité calée à droite*

Ainsi dans le cas où l'activité l est calée à gauche, nous avons :

$$overlap(IT,k,l) = min\{w_{kl} - (r_i - r_l) \times a_{lk}, (d_j - r_l) \times a_{lk}, w_{lk}\}$$

et l'ajustement :

$$slack(IT,k) < overlap(IT,k,l) \Rightarrow r_l := max\{r_l, d_j - \lfloor \frac{slack(IT,k)}{a_{lk}} \rfloor\}$$

Dans le cas où l'activité l est calée à droite, nous avons (nous rappelons que $d_f = f_l + p_l$) :

$$overlap(IT,k,l) = min\{(d_l - r_i) \times a_{lk}, w_{kl} - (d_l - d_j) \times a_{lk}, w_{lk}\}$$

et l'ajustement :

$$slack(IT,k) < overlap(IT,k,l) \Rightarrow d_l := min\{d_l, r_i + \lfloor \frac{slack(IT,k)}{a_{lk}} \rfloor\}$$

• **Règles de rejet (*reject rule*) :** Ces règles sont similaires à l'*edge-finding* pour les problèmes purement disjonctifs. Elles détectent si une activité doit être placée avant ou après toutes les activités de l'*intervalle de tâches*.

Nous avons les deux règles suivantes :

– $\forall l' \in IT \ (d_{l'} \leq r_l \ ou \ l' \rightarrow l) \Rightarrow l \ est \ après \ IT$

et l'ajustement :

$$r_l := max\{r_l, \ r_i + \lfloor \frac{energy(IT,k)}{R_k} \rfloor\}$$

– $\forall l' \in IT \ (d_l \leq r_{l'} \ ou \ l \rightarrow l') \Rightarrow l \ est \ avant \ IT$

et l'ajustement :

$$d_l := min\{d_l, \ d_j - \lfloor \frac{energy(IT,k)}{R_k} \rfloor\}$$

Dans [Caseau et Laburthe, 1996b], les auteurs ont de plus introduit la **règle d'exclusion** (*exclusion rule*) qui exprime le fait que si la somme des besoins en ressource de deux activités dépasse la capacité de la ressource alors ces activités sont en disjonction.

$$a_{lk} + a_{l'k} > R_k \Rightarrow l \leftrightarrow l'$$

2.5.3 Le raisonnement énergétique

Erschler et al. [1990] et *Lopez et al.* [1992] ont défini le concept de la quantité d'énergie exigée par une activité i sur un intervalle $I = [a,b]$. Notons par D_i^I la durée minimale durant laquelle l'activité i intersecte l'intervalle I. Il existe quatre possibilités (*Fig.* 2.7).

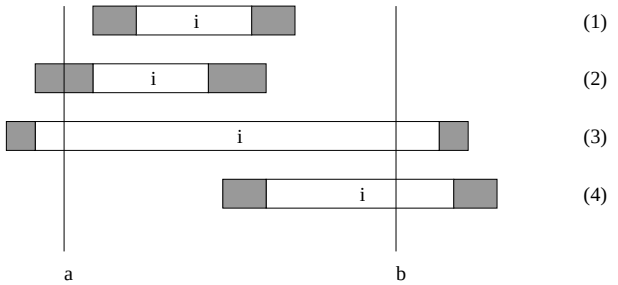


FIG. 2.7 – *Le raisonnement énergétique : cas possibles*

– $D_i^I := p_i$ dans le cas (1)

- $D_i^I := r_i + p_i - a$ dans le cas (2)
- $D_i^I := b - a$ dans le cas (3)
- $D_i^I := b - f_i$ dans le cas (4)

Ces quatre cas ainsi que le cas où l'activité i n'intersecte pas l'intervalle $[a, b]$ peuvent être résumés par la formule suivante :

$$D_i^I := max\{0, min\{p_i, r_i + p_i - a, b - a, b - f_i\}\}$$

La notion de raisonnement énergétique a été étendue aux problèmes d'ordonnancement avec contraintes de ressources. Elle est la technique la plus utilisée pour maintenir les contraintes de ressources. Son principe est le suivant :

Considérons un intervalle $[u, v]$. Pour chaque activité i, nous évaluons sa quantité de travail minimale (une unité de travail = une unité de temps $\times$ une unité de ressource) qui sera nécessairement occupée entre u et v. Nous fixons la date de début (*resp.* de fin) d'une activité à un instant t. Si nous obtenons une contradiction alors nous pouvons ajuster la date de début au plus tôt (*resp.* au plus tard) de cette activité.

Exemple 1 (Raisonnement énergétique) :
Dans la figure 2.8, nous avons une ressource de capacité 10 et trois activités 1, 2 et 3 de durées opératoires 10, 6 et 6, et ayant un besoin en ressources de 7, 6 et 5 respectivement. La quantité de travail disponible sur l'intervalle $[8, 12]$ est $(12 - 8) \times 10 = 40$, la quantité de travail minimale qu'occupe l'activité 2 entre 8 et 12 est $(12 - 8) \times 6 = 24$, il nous reste donc une quantité de travail égale à 16. Si nous testons l'hypothèse $t_1 = 8$, nous avons une contradiction car nous dépassons la quantité de travail restante $4 \times 7 = 28 > 16$ et donc nous ajustons la date de début au plus tôt de l'activité 1 à $r_1 := r_1 + \lceil \frac{28-16}{7} \rceil = 10$.
De même l'hypothèse $C_3 = 6$ nous conduit à une contradiction, nous avons donc l'ajustement $f_3 := f_3 - \lceil \frac{20-16}{5} \rceil = 11$.

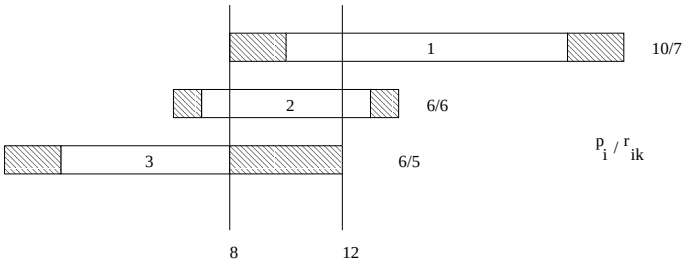


FIG. 2.8 – *Le raisonnement énergétique : un exemple*

Le raisonnement énergétique est efficace, mais il ne peut pas être applicable à tous les intervalles de temps. Ce qui nous oblige à réduire le nombre d'intervalles à tester. Mais alors deux difficultés se présentent :

- sur quel intervalle de temps appliquer le raisonnement énergétique?

– comment choisir les hypothèses à tester?

Baptiste et al. [1999] ont proposé un algorithme quadratique permettant de calculer l'énergie nécessairement consommé W pour des intervalles de temps pour lesquels l'énergie restant est minimal. L'algorithme commence par calculer toutes les valeurs prises par W durant les intervalles de temps $[t_1, t_2]$, où t_1 est un élément de l'ensemble $O_1 = \{r_i, 1 \leq i \leq n\} \cup \{f_i, 1 \leq i \leq n\} \cup \{r_i + p_i, 1 \leq i \leq n\}$, et par suite calculer toutes les valeurs prises par W durant les intervalles de temps $[t_1, t_2]$, où t_2 est un élément de l'ensemble $O_2 = \{d_i, 1 \leq i \leq n\} \cup \{r_i + p_i, 1 \leq i \leq n\} \cup \{f_i, 1 \leq i \leq n\}$. Pour cela, ils considérent la fonction $t \rightarrow W(i, t_i, t)$ qui est une fonction linéaire sur les intervalles délimités par les valeurs d_i, $r_i + p_i$, f_i et $r_i + d_i - t_i$. Cette propriété leur a permis de maintenir dans un mode incrémental l'accroissement de la fonction W.

Lopez et al. [1992] ont montré qu'appliquer le raisonnement énergétique sur un nombre quadratique d'intervalles de la forme $[r_i, d_j]$ est un bon compromis entre temps de calcul et efficacité.

2.5.4 Histogramme

Caseau et Laburthe [1996b] ont introduit un histogramme *time-table* représentant à chaque instant t le niveau de consommation atteint *level(t)*. Cet histogramme est utilisé pour :

– détecter une incohérence :

$$Si\ à\ un\ instant\ t,\ level(t) > R_k\ alors\ nous\ avons\ une\ contradiction$$

– ajuster les fenêtres temporelles des activités :
Tous les instants t pour lesquels $R_k - level(t) > a_{lk}$ peuvent être retirés de la fenêtre temporelle de l'activité l.

2.5.5 Parties obligatoires

Nous rappelons que la partie obligatoire (*core-times*) *PO(i)* associée à une activité i, est la longueur de l'intervalle durant lequel une partie de l'activité i est exécutée quelle que soit sa date de démarrage dans $[r_i, f_i]$. Nous avons vu précédemment (*cf.* section 2.2.2 page 9) que ces parties obligatoires peuvent permettre de calculer une borne inférieure de façon destructive. Elles peuvent également permettre de détecter des contradictions si un conflit de ressource apparaît en ordonnançant toutes les *PO(i)*. Sinon, pour chaque activité, une fenêtre temporelle durant laquelle cette activité peut être entièrement ordonnancée est recherchée. Dans le cas où il n'existe qu'un seul intervalle possible pour une activité i, sa fenêtre temporelle est alors réduite à cet intervalle et les fenêtres temporelles de ses successeurs et prédécesseurs sont ajustées. Ceci peut avoir pour effet de modifier les parties obligatoires. Il faut alors réitérer la procédure. Dans le cas où il n'existe pas d'intervalle, il y a contradiction.

2.6 Quelques extensions du RCPSP

Nous présentons, dans cette section, quelques extensions du RCPSP dont nous reparlerons dans la suite de ce mémoire. Bien que les extensions du RCPSP aient une grande importance dans la pratique, seulement quelques procédures de résolution exacte de ces extensions ont été présentées dans la littérature.

2.6.1 RCPSP avec relations de précédence généralisées

Dans le RCPSP, les activités sont reliées par des relations de précédence. Ces relations imposent à une activité j de commencer à n'importe quel instant après la fin d'une activité i. Ces relations peuvent être généralisées en imposant à l'activité j de commencer dans une certaine fenêtre de temps après la fin de l'activité i. Ces relations de précédence sont appelées : relations de précédence généralisées (*Generalized Precedence Relations* ou GPR).

Les relations de précédence généralisées peuvent être modélisées par des contraintes de la forme $C_i + l_{ij} \leq t_j$, où t_j est la date de début de l'activité j, $C_i = t_i + p_i$ la date de fin de l'activité i, et l_{ij} est un entier positif ou négatif. Lorsque $l_{ij} \geq 0$, la contrainte signifie que l'activité j ne peut démarrer avant que l_{ij} unités de temps se soient écoulées après la date de fin de l'activité i (*minimal time-lag*). Lorsque $l_{ij} < 0$, l'activité j peut débuter au plus tôt $(-l_{ij})$ unités de temps avant la date de fin de l'activité i (*maximal time-lag*). Notons que lorsque $l_{ij} = 0$, nous obtenons une contrainte de précédence classique.

Ce problème, noté RCPSP/max ou encore $m, 1 \mid gpr \mid C_{max}$ [Herroelen *et al.*, 1998], est *NP-difficile* et le problème d'existence d'une solution réalisable est *NP-complet* au sens fort [Bartusch *et al.*, 1988].

[Bartusch *et al.*, 1988, Reyck et Herroelen, 1998, Möhring *et al.*, 1998] ont développé des procédures par séparation et évaluation pour résoudre exactement le RCPSP/max. Leur idée commune consiste à calculer une solution optimale réalisable en relaxant toutes les contraintes de ressources. L'ordonnancement ainsi obtenu peut violer certaines contraintes de ressources. Un conflit de ressource est résolu en ajoutant de nouvelles contraintes qui retardent un ensemble d'activités qui causent ce conflit. Un nouvel ordonnancement est calculé, en prenant en compte ces nouvelles contraintes et en relaxant les contraintes de ressources. Cette procédure est répétée jusqu'à obtenir un ordonnancement sans conflit de ressources. Dans les algorithmes proposés par *Bartusch et al.* et *De Reyck et Herroelen*, les activités sont retardées en ajoutant des contraintes de précédence, tandis que dans l'algorithme proposé par *Möring et al.*, les conflits sont résolus en réduisant dynamiquement les dates de début au plus tôt de certaines activités.

Dans [Dorndorf *et al.*, 1998], les contraintes temporelles et de ressources sont considérées simultanément. La procédure consiste à énumérer les dates de début possibles d'une activité en se basant sur l'idée suivante : en un nœud donné de l'arbre de recherche, une activité doit commencer le plus tôt possible ou doit être retardée. Des techniques de propagation de contraintes sont de plus appliquées aux contraintes temporelles et de ressources au cours de la recherche dans le but de réduire l'ensemble des dates de début possibles de l'activité et donc de réduire l'espace de recherche.

2.6.2 Le RCPSP généralisé

Dans les contraintes de ressources, définies précédement, la capacité de chacune des ressources reste la même en chaque période de temps. Or, dans certaines applications réelles, elle peut varier au cours du temps.

Le RCPSP généralisé (*Generalized Resource Constrained Project Scheduling* ou GRCPSP) est une extension du RCPSP dans laquelle les contraintes temporelles sont définies par des relations de type *minimal time-lag*, à chaque activité est associée une fenêtre temporelle, et la disponibilité des ressources peut varier au cours du temps.

Le GRCPSP, noté aussi $m, 1, va \mid min, p_j, \delta_j \mid C_{max}$ [Herroelen *et al.*, 1998], est *NP-difficile*

et le problème d'existence d'une solution réalisable est *NP-complet* [Bartusch *et al.*, 1988].

Klein et Scholl [2000] ont développé une procédure exacte, appelée *Progress*, pour le GRCPSP. Les auteurs ont proposé une méthode par séparation et évaluation combinant un schéma de branchement parallèle et un nouveau type de stratégie *DFS* (*depth-first search*). Leur idée consiste à décomposer l'arbre de recherche en différents sous-arbres disjoints, à évaluer indépendamment chacun de ces sous-arbres et à considérer en premier les sous-arbres les plus prometteurs. En plus, selon le développement du processus de recherche, la recherche peut être diversifiée en consirérant un autre sous-arbre ou intensifiée en continuant à examiner le sous-arbre courant.

2.7 Conclusion

Le but de ce chapitre était de présenter les principales méthodes utilisées pour la résolution du RCPSP.

Nous avons présenté une variété d'approches existantes pour le calcul des bornes inférieures. Parmi ces approches, les plus performantes sont celles utilisant la méthode déstructive. Leur défaut est qu'elles sont très coûteuses en temps de calcul (citons par exemple la méthode proposée par *Brucker et Kunst* [2000] qui utilise des techniques de génération de colonne complexes).

Concernant la borne supérieure, les méthodes les plus utilisées sont celles basées sur les règles de priorité. En effet, leur mise en œuvre est très simple et ne nécessite pas beaucoup de temps de calcul. Ces méthodes sont plus au moins performantes. Quelques méthodes plus performantes, comme les méta-heuristiques, permettent un bon compromis entre la qualité des solutions trouvées et le temps de calcul.

En ce qui concerne les méthodes de résolution exactes, nous pouvons retenir les deux méthodes suivantes : celle proposée par *Demeulemeester et Herroelen* [1992] et la méthode proposée par *Brucker et al.* [1998]. La méthode proposée par *Demeulemeester et Herroelen* a donné de bons résultats. Son efficacité est liée a l'utilisation de la règle de dominance *Cut-Set*, mais cette règle nécessite une place-mémoire importante. Par contre, la méthode proposée par *Brucker et al.* est un bon compromis entre efficacité et place-mémoire utilisée. Cette méthode est moins performante pour les problèmes fortement cumulatifs, *i.e.* les problèmes pour lesquels les consommations en ressources sont faibles par rapport aux disponibilités des ressources.

Concernant les règles de déduction, plusieurs règles ont été proposées. Elles permettent de détecter des conflits de ressources et d'ajuster les fenêtres temporelles des activités. Nous avons des règles efficaces mais très coûteuses (comme les conditions nécessaires d'existence d'une solution et le raisonnement énergétique), et d'autres moins efficaces et moins coûteuses (comme l'histogramme et les intervalles de tâches). *Baptiste et Le Pape* [1997] ont montré experimentalement que l'utilisation des outils tels que les conditions nécessaires d'existence et les ajustements basés sur le raisonnement énergétique sont indispensables pour la résolution des problèmes fortement cumulatifs, mais moins efficaces pour les autres problèmes. *Caseau et Laburthe* [1994] ont utilisé la notion d'histogramme et des règles basées sur les intervalles de tâches pour la résolution du RCPSP. Les résultats ainsi obtenus sont très satisfaisantes.

En conclusion, nous pouvons constater qu'il existe un nombre assez important de techniques pour la résolution du RCPSP. Nous avons des méthodes approchées simples à mettre en œuvre, mais dont les performances sont assez faibles. D'autres méthodes approchées plus

compliquées permettant le plus souvent un bon compromis entre la qualité des solutions fournies et le temps de calcul. Nous avons aussi des méthodes exactes efficaces pour la résolution des problèmes de taille raisonnable (jusqu'à 30 activités).

Chapitre 3

Problèmes d'ordonnancement dynamiques : présentation et état de l'art

Sommaire

Introduction

Dans le chapitre 2, nous avons présenté les différentes approches proposées dans la littérature pour la résolution du RCPSP dans le cadre statique, où toutes les activités et toutes les contraintes sont connues à l'avance et restent inchangés durant toute la période d'exécution du système. Ces approches ont la particularité de fournir des algorithmes produisant un seul ordonnancement qui est fixé pendant toute la durée de la réalisation du système. Or, bien souvent, les systèmes considérés évoluent dans un environnement incertain ou perturbé, ce qui implique que l'ordonnancement se déroule rarement comme prévu.

Ce chapitre présente dans un premier temps ce qu'est un problème d'ordonnancement dynamique et la problématique associée. Nous présentons ensuite un état de l'art des principaux travaux qui ont été développés pour les problèmes d'ordonnancement en présence de perturbations. Nous avons classé ces travaux selon la terminologie proposée par *Davenport et Beck* [2000] à savoir : approches proactives, approches réactives et approches proactives-réactives. Un état de l'art de ces approches peut être trouvé dans [flexibilité du Gotha, 2002] et [Davenport et Beck, 2000]. Nous introduisons ensuite quelques mesures de performance des approches rencontrées dans la littérature. En conclusion, nous situons notre approche par rapport aux approches présentées.

3.1 Problématique

Un problème d'ordonnancement dynamique peut être vu comme un problème qui évolue au cours du temps. Cette évolution est causée par l'apparition d'aléas imprévus, changeant ainsi les données du problème. Parmi ces aléas, nous pouvons citer par exemple : de nouvelles commandes arrivent en cours de production, d'autres sont supprimées, les matières premières ne sont pas livrées à temps, les quantités produites ne sont pas les bonnes, les machines utilisées ne sont pas celles prévues, une coupure de courant survient à un moment critique de la production, les réglages des machines prennent plus de temps que prévu, les machines tombent en panne, les réparations des machines ne sont pas faites correctement, l'ordre de passage de certaines activités est modifié, ...

Les différents aléas que peut subir un système de production peuvent être regroupés en trois catégories : des aléas portant sur les activités, des aléas portant sur les ressources et des aléas temporels. Le tableau 3.1 fournit quelques exemples d'aléas pour chacune de ces catégories.

Notons que certains de ces aléas (comme le retrait de certaines ressources, le retrait de certaines activités ou le retrait de certaines contraintes temporelles) peuvent être ignorés lorsque nous souhaitons avoir une solution réalisable. Par contre l'ordonnancement doit être ré-optimisé si l'on souhaite une solution optimale.

La présence de ces aléas fait que les ordonnancements calculés ne restent pas réalisables ou valides très longtemps. Ces ordonnancements ont donc besoin d'être ré-actualisés régulièrement. En effet, devant la forte compétition existante aujourd'hui (qui entraîne de plus en plus de perturbations au cours de la production), les décideurs ont besoin de systèmes capables de réagir vite aux perturbations, et fournissant des ordonnancements performants.

Se pose alors le problème de la performance des systèmes d'ordonnancement. Plusieurs mesures ont été introduites. Nous parlons de la robustesse d'un algorithme ou d'un système lorsque celui ci est capable de produire des ordonnancements en garantissant une certaine per-

Catégories	Exemples
Aléas portant sur les activités	- ajout ou retrait d'activités - augmentation ou diminution de la durée opératoire de certaines activités - augmentation ou diminution des quantités de ressources nécessaire pour l'exécution de certaines activités - modification de la date de disponibilité de certaines activités - modification de la date échue de certaines activités
Aléas portant sur les ressources	- augmentation ou diminution du nombre de ressources disponibles (achat de nouvelles machines, licenciement) - augmentation ou diminution de la capacité d'une ressource durant une période de temps (panne de machines ou retour de machines qui étaient en maintenance)
Aléas temporels	- modification de l'ordre d'exécution d'un ensemble d'activités (ajout ou retrait de précédences, de disjonctions ...)

TAB. 3.1 – *Classification des différents aléas par catégories*

formance. La stabilité est définie comme une propriété liant les solutions successives produites par un algorithme ou un système d'ordonnancement. Une solution est stable par rapport à la précédente si le passage d'une solution à l'autre se fait par de petits changements.

Nous présentons dans les sections suivantes les différentes approches proposées pour résoudre ces problèmes, ainsi que les différentes mesures de performance utilisées.

3.2 Approches de résolution

Dans la littérature, plusieurs approches ont été proposées pour faire face aux aléas (données incertaines et perturbations) que peut subir un système d'ordonnancement. *Mehta et Uzsoy* [1999] et *Davenport et Beck* [2000] ont proposés deux classifications différentes de ces approches.

La classification proposée par *Mehta et Uzsoy* regroupe quatres catégories d'approches : des approches totalement réactives, des approches prédictives réactives, des approches robustes et enfin des approches à base de connaissances. Quant à la classification proposée par *Davenport et Beck*, elle regroupe les trois catégories d'approches suivantes : des approches réactives, des approches proactives et des approches proactives réactives.

Nous avons choisi de présenter les principaux travaux développés pour la résolution des problèmes d'ordonnancement en présence de perturbations selon la classification proposée par *Davenport et Beck* car elle permet de couvrir toutes les approches existantes. En effet, les approches proactives réactives (utilisant deux algorithmes un proactif et l'autre réactif) ne peuvent pas être classées lorsque nous considérons la terminologie de *Mehta et Uzsoy*.

3.2.1 Approches proactives

Ces approches visent à construire grâce à un algorithme robuste une solution statique dite flexible (*i.e.* garantissant un certain niveau de performance pour un ensemble de données possibles), telle que peu de modifications soient nécessaires en cas de perturbations des données

initiales au cours de son exécution.

Parmi ces méthodes, certaines consistent à laisser volontairement des temps morts sur les machines de façon à pouvoir facilement traiter des activités inattendues ou à pouvoir transférer des activités sur d'autres machines en cas de panne, tout en maintenant un niveau de performance donné. Cette approche est par exemple utilisée par *Mehta et Uzsoy* [1999] pour un problème à une machine avec des arrivées dynamiques d'activités et des pannes de machines aléatoires. L'objectif considéré est de minimiser le retard maximum des activités. La méthode proposée est reprise dans [O'Donovan *et al.*, 1999] pour le même problème en prenant comme critère le retard moyen des activités, et étendue pour tenir compte d'activités dites sensibles (dont la durée opératoire augmente en cas de panne de machine).

D'autres se basent sur des raisonnements probabilistes. Elles ont pour objectif de construire un ordonnancement tel que la probabilité d'atteindre une certaine performance soit la plus grande possible. Par exemple, *Daniels et al.* [1997] proposent une telle approche dans le cas d'un problème à une machine avec des durées opératoires incertaines. Le critère considéré est le temps de séjour total des activités dans le système. Étant donné un ensemble de scénarios (valeurs possibles des durées opératoires), les auteurs proposent une recherche arborescente et une heuristique construisant un ordonnancement maximisant la probabilité d'atteindre un certain niveau de performance.

D'autres types d'approches ont été proposées. Par exemple dans [Sevaux et Sörensen, 2002] et [Sörensen, 2001], les auteurs s'intéressent à l'ordonnancement de pièces à assembler sachant que les dates de livraison de ces pièces sont incertaines, avec comme critère le nombre pondéré d'activités en retard. Un ordonnancement robuste est obtenu par un algorithme génétique dans lequel, à chaque itération, le critère est évalué pour une série d'instances dont les dates de disponibilité ont été modifiées aléatoirement.

Ces approches sont adaptées uniquement lorsque l'on sait à l'avance quelles sont les aléas qui risquent d'intervenir et ne garantissent une solution robuste que dans le cas de petits changements des données initiales. De plus, ces approches ne garantissent pas l'optimalité. Enfin, on notera que la plupart de ces travaux concernent les problèmes à une machine, et que les problèmes plus complexes comme les problèmes d'ateliers ou le RCPSP sont peu étudiés.

3.2.2 Approches réactives

Ces approches sont utilisées au moment de la réalisation de l'ordonnancement en tenant compte de l'état courant du système.

Dans certaines de ces approches, aucune solution n'est proposée au préalable. L'ordonnancement est construit au fur et à mesure des besoins. Ces approches, dites *en ligne* (ou *online*) sont utilisées lorsque les perturbations sont courantes ou connues très tardivement, rendant ainsi inutile la construction d'un ordonnancement prédictif. Les méthodes utilisées sont généralement de simples règles de priorité qui consistent, quand une ressource devient disponible, à sélectionner une activité dans la file d'attente suivant un certain critère. Un état de l'art sur ces approches est présenté dans [Sabuncuoglu et Bayiz, 2000].

Cette approche a l'inconvénient de rendre difficile la planification d'autres activités relatives à l'ordonnancement comme l'approvisionnement en matières premières ou le planning du personnel. De plus, elle ne garantit pas une solution de bonne qualité. Néanmoins, elle est très utilisée lorsque les perturbations sont très complexes.

D'autres méthodes se basent sur un ordonnancement prédictif calculé au préalable et réordonnancent cette solution quand des aléas interviennent. L'application de ces méthodes

sucite les deux questions suivantes : Quand doit-on ré-ordonnancer ? et Comment doit-on ré-ordonnancer ?

3.2.2.1 Quand ré-ordonnancer ?

Nous pouvons choisir de ré-ordonnancer à chaque fois qu'un événement inattendu apparaît. C'est le cas dans [Wu *et al.*, 1993] où les auteurs proposent une heuristique pour ré-ordonnancer un problème à une machine en cherchant à maximiser la stabilité. Et également dans [Bierwirth et Mattfeld, 1999], où les auteurs présentent un algorithme génétique réutilisant la solution courante pour résoudre un problème de *Job-Shop* chaque fois qu'un nouveau job arrive. L'inconvénient de cette approche est que les ré-ordonnancements risquent d'être nombreux, provoquant ainsi une instabilité des ordonnancements calculés et nécessitant des temps de calcul élevés.

Nous pouvons aussi choisir de ré-ordonnancer à des intervalles de temps réguliers. Mais alors l'ordonnancement est moins performant que dans la première solution. Cette approche est utilisée dans les travaux de *Church et Uzsoy* [1992] : les auteurs ont développé une technique de ré-ordonnancement périodique pour les problèmes à une machine et à machines parallèles avec des arrivées dynamiques d'activités. Dans cette approche, les événements inattendus apparaissant entre les périodes de ré-ordonnancement sont ignorés, sauf s'il s'agit d'événements importants nécessitant des procédures de ré-ordonnancement d'urgence.

Enfin, dans [Vieira *et al.*, 2000], les auteurs étudient le problème à une machine dans lequel les activités arrivent dynamiquement. Leurs proposition consiste à ré-ordonnancer dès que le nombre d'arrivées de jobs atteint un seuil donné. Ils montrent que ré-ordonnancer fréquemment permet d'obtenir de meilleurs performances du système mais entraîne une augmentation du nombre de réglages, tandis que des ré-ordonnancements moins fréquents augmentent les temps de cycles.

En général, des ré-ordonnancements fréquents permettent d'obtenir de meilleures performances parce que le système réagit rapidement à des événements non planifiés. Cependant, ceci peut demander beaucoup d'effort de calcul et peut conduire à de nombreux temps de réglages des machines.

3.2.2.2 Comment ré-ordonnancer ?

Deux alternatives sont considérées : nous nous contentons d'avoir une nouvelle solution réalisable, dans ce cas nous parlons de *réparation*, ou la nouvelle solution doit être optimale, nous parlons alors de *post-optimisation* (ou *ré-optimisation*).

• **Réparation :** Plutôt que de ré-optimiser le système, les décideurs se contentent souvent de réparer la solution. Dans ce cas, l'optimalité n'est pas garantie. Cette technique est souvent utilisée par les personnes en charge de l'ordonnancement qui peuvent ainsi utiliser leur expérience et leur connaissance du problème pour déplacer des activités ou ré-affecter des activités sur d'autres machines en cas de panne. Une méthode simple consiste, en cas de panne de machines, à conserver l'ordre d'exécution des activités et à repousser l'exécution des activités affectées par la panne. Nous pouvons aussi utiliser d'autres règles simples (basées sur les durées opératoires, les dates échues, ...) permettant de trier les activités selon un critère donné et de les affecter aux machines dans cet ordre. Ce genre de règle simple n'est généralement pas très performant. Des techniques plus efficaces (mais demandant plus d'effort de calcul) ont été développées.

Ainsi, dans [Bean *et al.*, 1991], les auteurs proposent une procédure réparant l'ordonnancement prédictif quand des événements inattendus surviennent de telle sorte que le nouvel ordonnancement recouvre autant que possible l'ordonnancement initial. Les auteurs montrent que cette technique est plus avantageuse que les règles simples vues plus haut. Cette approche a été appliquée ensuite dans [Akturk et Gorgulu, 1999] au problème de *Flow-Shop* modifié soumis à des pannes de machines. Dans cet article, les auteurs montrent que cette technique est très efficace en terme de qualité des ordonnancements calculés, de temps d'exécution et de stabilité des solutions.

Dans [Bierwirth et Mattfeld, 1999], les auteurs présentent un algorithme génétique utilisé dans un premier temps pour construire un ordonnancement prédictif, puis ensuite pour réagir aux arrivées aléatoires d'activités. L'idée est alors de partir de la population résultant de la dernière exécution de l'algorithme génétique pour trouver une nouvelle solution incluant ces nouvelles activités. Cette technique permet une convergence plus rapide et donc une économie en temps d'exécution.

Dans [Wu *et al.*, 1993], les auteurs proposent une heuristique de ré-ordonnancement multicritère pour un problème à une machine. Ils associent un coût de retard, un coût d'avance et un coût de changement de séquence aux modifications de l'ordonnancement et utilisent comme fonction objectif une somme pondérée du makespan et du coût total de ré-ordonnancement. Les auteurs utilisent également différentes recherches locales basées sur des algorithmes génétiques.

Il existe aussi dans la littérature des approches basées sur l'intelligence artificielle. ISIS [Fox, 1987], OPIS [Smith *et al.*, 1990] et IOSS [Park *et al.*, 1996] sont des systèmes à base de connaissance générant des ordonnancements réalisables et proposant des méthodes intéractives pour modifier un ordonnancement existant. Par exemple le système ISIS répare l'ordonnancement en donnant la priorité à la solution dans laquelle un maximum de variables gardent leur ancienne valeur. CABINS [Miyashita, 1995] est un système d'ordonnancement intelligent permettant de réutiliser des expériences passées pour réparer l'ordonnancement. Un résumé de ces différentes approches peut être trouvé dans [Szelke et Kerr, 1994].

• **Post-optimisation :** La ré-optimisation est peu appliquée. [Picouleau, 2000] a montré que, pour les problèmes à 1, 2 ou m machines (qui sont *NP-difficiles*) et pour lesquels nous avons introduit une perturbation même minimale (augmentation ou diminution d'une date de disponibilité, d'échéance ou d'une durée opératoire), le problème de *post-optimisation* est aussi *NP-difficile*.

Dans le cas extrême, la ré-optimisation consiste à recalculer un ordonnancement à partir de zéro quand un aléa intervient. Ceci risque de génèrer des ordonnancements très différents les uns des autres et les temps de calculs risquent d'être prohibitifs.

Nous n'avons pas rencontré de telles approches dans la littérature.

3.2.3 Approches proactives-réactives

Ces techniques sont constituées de deux phases : dans une première phase un ordonnancement prédictif robuste tenant compte de l'aspect incertain de certaines données est construit par un algorithme dit *statique* ou *hors-ligne*; cet ordonnancement est ensuite adapté en fonction de l'état du système au moment de son exécution grâce à un algorithme dit *dynamique* (on parle d'algorithme *en-ligne*).

La plupart de ces méthodes construisent un ensemble d'ordonnancements statiques tels qu'il soit facile de passer de l'un à l'autre en cas d'aléa. Elles se basent sur la recherche de

groupes de tâches permutables sur chaque ressource, *i.e.* toutes les tâches d'un même groupe sont totalement permutables sans diminuer la performance souhaitée. Le résultat est donc un ensemble d'ordonnancements obtenus en énumérant toutes les permutations possibles au sein de chaque groupe, parmi lesquels le décideur peut choisir en temps réel celui qu'il souhaite mettre en place en fonction de ses préférences, des contraintes non modélisées, ou des aléas.

La méthode ORABAID (*ORdonnacement d'Atelier Basé sur l'AIde à la Décision*), conçue par une équipe de LAAS (*Laboratoire d'Analyse et d'Architecture des Systèmes*) de Toulouse, utilise une telle approche [Demmou, 1977].

Nous retrouvons aussi ce type d'approche dans [Wu *et al.*, 1999]. Les auteurs cherchent à minimiser la somme pondérée des retards dans un problème de *Job-Shop*. Ils présentent une méthode par séparation et évaluation (Branch-and-Bound) permettant de calculer une sorte de squelette de l'ordonnancement donnant une vue globale du système. Ce squelette est ensuite complété au moment de son exécution par des décisions prises dynamiquement suivant les perturbations qui peuvent se produire.

Dans [Esswein *et al.*, 2002] et [Esswein *et al.*, 2003], les auteurs proposent une telle approche en utilisant une méthode de programmation dynamique permettant d'obtenir un ensemble d'ordonnancements dont la date de fin est inférieure ou égale à une borne supérieure donnée.

Aloulou et Portmann [2002] considèrent un problème à une machine avec comme fonctions objectifs la somme pondérée des retards ou la date de fin de l'ordonnancement. Ils proposent un algorithme génétique construisant plusieurs ordonnancements équivalents (*i.e.* respectant un ordre partiel des activités) et garantissant un bon compromis entre flexibilité et performance. Ensuite, à chaque fois qu'une décision doit être prise suite à un aléa (retards de livraison de matières premières, retards d'exécution des activités et pannes de machines), plusieurs alternatives sont proposées au décideur.

Moukrim et al. [2003] ont considéré le problème d'ordonnacement sur m processeurs identiques avec temps de communication. Le temps de communication entre deux activités n'est pris en compte que lorsque ces deux activités sont liées par une relation de précédence et qu'elles doivent être exécutées sur deux processeurs différents. L'objectif est de minimiser la durée totale de l'application. Les auteurs ont proposé une approche composée de trois étapes : la premiére calcule un ordonnancement prédictif en utilisant une heuristique de priorité. La seconde consiste à ajouter des précédences entre les activités affectées à un même processeur. Ensuite, dans une troisième étape, l'heuristique de la première phase est utilisée pendant le déroulement de l'ordonnancement pour compléter les décisions non encore prises. Les auteurs ont montré d'une part que leur approche est optimale pour deux types de graphes et d'autre part qu'elle est meilleure par rapport à une approche complètement réactive pour des graphes quelconques.

Artigues et al. [2002] ont proposé un algorithme polynomial permettant d'insérer une activité imprévue dans un RCPSP. Le principe est que les ressources sont considérées comme des flots et que toute unité de ressource utilisée pendant la réalisation d'une activité est transférée vers une unique autre activité. Étant donné un graphe représentant un ensemble d'ordonnancements et une activité à inclure dans ce graphe selon des contraintes de succession dans le projet et d'utilisation des ressources déterminées, l'algorithme polynomial présenté cherche à trouver une position d'insertion de cette activité dans le graphe en minimisant la durée totale du projet sans modifier les flots existants. Dans le but de ne pas trop perturber la solution précalculée, les auteurs imposent à leur algorithme de conserver le séquencement des activités déjà présentes et essaient d'insérer l'activité tout en minimisant la durée totale du projet.

Leur algorithme applique ensuite une méthode sérielle pour rendre actif l'ordonnancement ainsi obtenu. D'après leur expérimentations, les auteurs ont montré que leur méthode est, en général, plus robuste que le réordonnancement.

Dans [Artigues, 2003], l'auteur s'est interéssé aux deux problèmes d'ordonnancement : le problème à une machine et le *Job-Shop*. Pour chacun de ces problèmes, un algorithme polynomial permettant de maximiser la flexibilié dans un ordonnancement robuste a été présenté. L'approche considérée pour déterminer un ordonnancement robuste consiste à calculer plusieurs ordonnancements réalisables ou optimaux. Ainsi, pour le problème de *Job-Shop*, l'auteur utilise l'approche proposée par *Erschler et Roubellat* [1989] qui consiste à produire un ensemble de groupes d'activités liées par des relations de précédence dans le but de déterminer un ordonnancement robuste. En étendant un résultat obtenu dans [Mauguiére *et al.*, 2002], l'auteur a développé un algorithme polynomial pour réduire le nombre de groupes en ne gardant que ceux qui conduisent à au moins un ordonnancement réalisable. Cette flexibilité est alors exploitée pour réagir aux perturbations en temps réel. Concernant les problèmes à une machine, l'auteur a introduit deux objectifs représentant la flexibilité d'un groupe d'activités. Le premier est de maximiser le nombre d'orientations des disjonctions associées à un groupe et qui sont réalisables et le second est de minimiser le nombre de groupes. Pour le second critère, il a proposé un algorithme polynomial permettant de construire un groupe optimal représentant une séquence d'activités donnée. Par contre pour le premier critère, l'auteur a introduit un algorithme polynomial lorsqu'une séquence d'activtités donnée est representée. Un contre-exemple a été fourni pour montrer que ce dernier algorithme ne donne pas nécessairement une solution optimale.

Le principal inconvénient de ces méthodes est que nous ne sommes pas sûrs de trouver parmi les ordonnancements générés dans la première phase une solution cohérente avec les aléas qui se présenteront.

3.3 Mesures de performance

Dans les sections précédentes nous avons présenté les principles approches qui ont été développées pour résoudre les problèmes d'ordonnancement en présence de perturbations. Néanmoins, ces approches ont le même but : celui de produire des ordonnancements en un temps raisonnable et avec un certain degré de performance.

Dans la littérature, nous distinguons deux critères permettant de mesurer la performance de ces approches : la *robustesse* et la *stabilité*.

3.3.1 Robustesse

Ces mesures sont utilisées dans le but de produire un ordonnancement prédictif initial qui tient compte des perturbations que peut subir le système. Le but est de minimiser une mesure de déviation entre les performances envisagées et les performances réalisées.

Ainsi, *Leon et al.* [1994] ont étudié le problème de *Job-Shop* en présence d'aléas liés à des pannes de machines. Leur but est de construire un ordonnancement initial robuste. Un algorithme génétique est utilisé pour trouver une meilleure solution minimisant la durée totale de l'ordonnancement et la déviation par rapport à la durée totale de l'ordonnancement initial. Les auteurs ont défini une mesure de robustesse comme étant la combinaison linéaire de l'espérance mathématique de la durée totale et de l'espérance mathématique de l'augmentation algébrique de la durée totale (déclarée comme une variable aléatoire). Pour calculer cette

mesure, qui est très difficile lorsque plusieurs pannes se présentent, les auteurs ont proposé plusieurs expressions rapprochant la valeur de l'espérance mathématique de l'augmentation de la durée totale. Ils ont montré expérimentalement que l'expression rapprochant le mieux cette espérance est celle basée sur les marges (la durée pour laquelle l'activité peut être retardée sans augmenter la durée totale de l'ordonnancement et sans changer l'ordre d'exécution des activités sur les machines).

Mehta et Uzsoy [1999] considérent le problème sur une machine avec arrivée dynamique des activités et panne de machines et comme objectif la minimisation du plus grand retard algébrique. Pour cela, ils calculent une séquence d'activités initiale en utilisant l'heuristique *ATC* (*Apparent Tardiness Cost*) [Rachamadugu et Morton, 1982] et insèrent des temps morts entre les activités de façon à absorber les pannes sans affecter les activités planifiées tout en préservant de bonnes performances.

Plusieurs autres travaux sur les approches proactives ont permis de fournir des mesures de robustesse [Kouvelis et Yu, 1997, Sevaux et Sörensen, 2002]. Ces mesures sont spécifiques au problème traité et aux aléas étudiés. Certaines de ces mesures peuvent être étendues pour d'autres problèmes (*cf.* [flexibilité du Gotha, 2002]).

3.3.2 Stabilité

La stabilité est une propriété qui lie les solutions successives produites par un système d'ordonnancement dynamique. Une solution est stable, par rapport à la solution précédente, si le passage d'une solution à l'autre se fait par de petits changements.

Ainsi, *Wu et al.* [1993] ont étudié le problème de réordonnancement sur une machine. Leur objectif est de trouver une nouvelle solution qui optimise la durée totale et la stabilité de l'ordonnancement. Ils ont considéré deux critéres de stabilité : le premier est la déviation de l'ordonnancement modifié en terme de dates de début des activités, et le second est la déviation de l'ordonnancement modifié par rapport à l'ordonnancement original en terme de séquence des activités. La mesure associée au premier critère est facile à calculer, elle est égale à la moyenne de l'écart entre les dates de début des activités de l'ordonnancement modifié et celles de l'ordonnancement original. Par contre, le second critère est difficile à mesurer. Une des mesures possibles est de calculer la somme des déviations des dates de débuts des activités des deux ordonnancements une fois les activités calées à droite.

Cowling et Johansson [2002] ont décrit une approche générale pour mesurer la stabilité d'un système d'ordonnancement et présentent une mesure de stabilité spécifique qui calcule le changement moyen des dates de début et de fin de deux ordonnancements.

3.4 Conclusion

Dans ce chapitre, nous avons présenté différentes approches permettant de tenir compte des aléas dans des systèmes d'ordonnancement dynamiques. Toutes ces approches concernent des problèmes simples (à une machine pour la plupart) et sont adaptées à des perturbations bien particulières (par exemple arrivées de nouvelles activités, ou bien pannes de machines) mais rarement à des aléas quelconques. De plus, l'efficacité de ces approches dépend du niveau de connaissance de l'incertitude et des types d'aléas traités.

Les approches proactives sont envisageables lorsque nous sommes en présence d'aléas connus à l'avance. Ces approches garantissent un certain niveau de performance. Tandis que les approches réactives sont envisagées lorsque les perturbations sont complétement inconnues.

Le défaut de ces approches est que leur utilisation ne garantie aucun niveau de performance. Signalons que nous n'avons pas rencontré dans la littérature d'approche réactive permettant de produire, à chaque perturbation, une nouvelle solution optimale.

L'approche que nous présentons dans cette thèse est une approche réactive dans laquelle la solution est ré-optimisée à chaque fois qu'un aléa se présente. Elle nécessite dans un premier temps de construire un ordonnancement prédictif optimal. Elle est capable de réagir à n'importe lequel des aléas présentés dans le tableau 3.1 (*cf.* section 3.1 page 26) pour le RCPSP.

Nous nous sommes basés principalement sur les techniques existantes pour la résolution des problèmes de satisfaction de contraintes dans le cadre dynamique. Ces techniques sont présentées dans les chapitres 4 et 5.

Chapitre 4

Problèmes de Satisfaction de Contraintes (CSP)

Sommaire

Introduction

Le formalisme des problèmes de satisfaction de contraintes (*Constraint Satisfaction Problems* ou CSP) offre un cadre unifié pour modéliser, étudier et résoudre de nombreux problèmes combinatoires. En effet, la notion de contraintes est un concept très générique qui permet d'offrir une grande expressivité. Une contrainte est une relation *a priori* quelconque, définie par un sous-ensemble de variables du problèmes et un sous-ensemble du produit cartésien des domaines de valeurs autorisées pour ces variables. L'intérêt d'une telle représentation est de permettre le développement de méthodes génériques de résolution, et ce, indépendamment du problème traité.

Le modèle CSP est intrinsèquement statique : le problème est défini une fois pour toutes, l'ensemble des variables, leurs domaines, les contraintes les liant sont connus à l'avance et ne peuvent être modifiés pendant la résolution. Ainsi, rapidement le modèle montre ses limites, en particulier lorsque les problèmes à résoudre sont dynamiques (*i.e.* des aléas devant être impérativement pris en compte interviennent pendant la résolution) ou encore sur-contraints (*i.e.* ne possédant pas de solution).

Dans ce chapitre, nous introduisons le formalisme des CSP et traitons de leur résolution. Nous présentons ensuite les problématiques particulières liées à l'optimisation, la résolution de problèmes dynamiques et la résolution de problèmes sur-contraints.

4.1 Définitions

On se donne un **domaine de calcul** $\mathcal{D}$. Il s'agit d'un ensemble quelconque (fini ou infini, dénombrable ou indénombrable) de valeurs.

> **Exemple 2 (Domaine de calcul) :**
> Si les valeurs que l'on considère sont discrètes, on peut alors prendre $\mathcal{D} = \mathbb{N}$. Si les valeurs considérées sont continues, on peut prendre $\mathcal{D} = \mathbb{R}$ ou encore $\mathcal{D} = \mathbb{Q}$. Les valeurs peuvent très bien être symboliques. Ainsi on peut avoir $\mathcal{D} = \{\text{passable}, \text{assez bien}, \text{bien}, \text{très bien}\}$.

On se donne un ensemble $V = \{v_1, \ldots, v_n\}$ de n **variables** prenant chacune leurs valeurs dans l'ensemble $\mathcal{D}$. À chaque variable v de V est associé un ensemble D_v appelé **domaine de** v qui contient l'ensemble fini ou infini des valeurs possibles pour la variable, *i.e.* $D_v \subset \mathcal{D}$.

On appelle **instanciation** d'une variable l'affectation à cette variable d'une des valeurs de son domaine.

On se donne un ensemble C de e **contraintes** portant sur des éléments de V. Une contrainte est une relation devant être vérifiée par l'instanciation des variables concernées.

> **Exemple 3 (Contrainte) :**
> Soient deux variables x et y. $x^2 = y + 1$ est alors une contrainte. Si x vaut 2 et y vaut 3, la contrainte est vérifiée.
> Soient deux variables x et y. Alors c : $\{(1,2),(2,4),(4,8)\}$ est aussi une contrainte. On dit qu'elle est exprimée en extension. Soit $(a,b) \in c$ (dans le cas d'une contrainte exprimée en extension, on identifie la vérification d'une contrainte et l'appartenance à la relation – l'ensemble – la définissant). Si x vaut a et y vaut b alors la contrainte est vérifiée.

On note $var(c)$ l'ensemble ordonné des variables concernées par la contrainte c. Le cardinal de $var(c)$ est appelé l'**arité** de la contrainte c.

Définition 3 (Problème de satisfaction de contraintes)
*Un **problème de satisfaction de contraintes** est défini par un triplet (V, D, C), où V est un ensemble fini de variables, D est l'ensemble des domaines associés à ces variables (dans le domaine de calcul $\mathcal{D}$) et C est l'ensemble des contraintes qui portent sur ces variables.*

Exemple 4 (Problème de satisfaction de contraintes) :
Considérons un problème d'ordonnancement comportant quatre activités ayant chacune une date de démarrage au plus tôt et au plus tard (*cf.* Tableau 4.1). Ces activités sont liées par les contraintes de précédence : $1 \rightarrow 2$, $1 \rightarrow 3$, $2 \rightarrow 4$ et $3 \rightarrow 4$.
Notons que la contrainte de précédence $i \rightarrow j$ est équivalente à la contrainte $t_i + p_i \leq t_j$.
Ce problème peut être modélisé comme un CSP (V, D, C) avec :
$V = \{t_1, t_2, t_3, t_4\}$
$D = \{D_1 = [0, 7], D_2 = [2, 6], D_3 = [1, 10], D_4 = [4, 13]\}$
$C = \{c_1 : t_1 + 3 \leq t_2, c_2 : t_1 + 3 \leq t_3, c_3 : t_2 + 2 \leq t_4,$
$c_4 : t_3 + 4 \leq t_4\}$.

Activité	r_i	f_i	p_i
1	0	7	3
2	2	6	2
3	1	10	4
4	4	13	1

TAB. 4.1 – *Données de l'exemple 4*

Une instanciation $a = (a_1, a_2, ..., a_l)$ satisfait une contrainte c, portant sur le sous-ensemble de variables $var(c) = \{v_{c_1}, v_{c_2}, ..., v_{c_k}\}$, si et seulement si $a' = (a_{c_1}, a_{c_2}, ..., a_{c_k})$ la restriction de a à $var(c)$ vérifie la contrainte c.

Une instanciation $a = (a_1, a_2, ..., a_l)$ est dite cohérente si et seulement si elle satisfait toutes les contraintes de C.

Une **solution** d'un CSP est une instanciation cohérente de l'ensemble des variables du problème satisfaisant toutes les contraintes du problème.

4.2 Recherche de solution pour un CSP

Résoudre un CSP est un problème *NP-difficile*. La méthode classiquement adoptée pour résoudre un tel problème utilise un algorithme énumératif basé sur le retour arrière. Bien sûr, pour améliorer la résolution, des techniques de réduction de l'espace de recherche sont utilisées. Généralement, ces techniques ont pour but de réduire l'espace de recherche en éliminant de chaque domaine des variables des valeurs que l'on peut prouver comme n'appartenant à aucune des solutions réalisables du CSP. Les techniques principalement utilisées sont des algorithmes dits de *cohérence locale* qui permettent d'utiliser des conditions nécessaires d'existence de

solution pour éliminer *a priori* des portions conséquentes de l'espace de recherche. On peut appliquer ces techniques aussi bien avant que pendant la phase de recherche de solution.

4.2.1 Cohérences locales

On peut rappeler quelques cohérences locales parmi les plus connues et les plus utilisées.

4.2.1.1 Arc cohérence

L'*arc cohérence* (*Arc Consistency* ou AC) [Waltz, 1975] est la plus répandue des cohérences locales. Elle est définie pour les CSP binaires (chaque contrainte portent sur deux variables).

Définition 4 (Arc cohérence)
Soit v_i une variable et a_i une valeur de son domaine D_{v_i}.

On dit que le couple (v_i, a_i) vérifie l'arc cohérence si et seulement si pour chaque contrainte c telle que $v_i \in var(c)$ il est possible de trouver une instanciation réalisable des variables de la contrainte c pour laquelle la valeur a_i est affectée à la variable v_i. La valeur de l'autre variable (nous sommes dans un CSP binaire) dans cette instanciation est appelée un support de a_i pour c.

Un domaine D_{v_i} vérifie l'arc cohérence si et seulement si il est non vide et toutes ses valeurs vérifient l'arc cohérence.

Un CSP vérifie l'arc cohérence si et seulement si tous ses domaines vérifient l'arc cohérence.

En d'autres termes, le filtrage par arc cohérence consiste à éliminer les éléments qui ne peuvent pas localement (en se contentant d'observer la situation uniquement du point de vue de la contrainte traitée) figurer dans une solution. Notons qu'un CSP qui vérifie l'arc cohérence n'admet pas forcément une solution.

4.2.1.2 Une généralisation : la k-arc cohérence

La notion d'arc cohérence peut être étendue à k variables. On obtient alors la *k-arc cohérence*.

Soit $S = \{v_{i_1}, v_{i_2}, ..., v_{i_{k-1}}\}$ un ensemble de $k-1$ variables. Une instanciation I_S de S consiste à affecter à chaque variable v_{i_j} de S une valeur a_{i_j} de son domaine. I_S sera notée $\{(v_{i_1}, a_{i_1}), (v_{i_2}, a_{i_2})..., (v_{i_{k-1}}, a_{i-k-1})\}$.

Définition 5 (k-cohérence)
On dit que $I_S = \{(v_{i_1}, a_{i_1}), (v_{i_2}, a_{i_2})..., (v_{i_{k-1}}, a_{i_{k-1}})\}$ vérifie la k-cohérence si et seulement si pour toute variable supplémentaire $v_{i_k} \notin S$, il existe une valeur $a_{i_k} \in D_{v_{i_k}}$ telle que l'instanciation $\{(v_{i_1}, a_{i_1}), (v_{i_2}, a_{i_2})..., (v_{i_{k-1}}, a_{i_{k-1}}), (v_{i_k}, a_{i_k})\}$ soit cohérente.

Un sous-ensemble $S = \{v_{i_1}, v_{i_2}, ..., v_{i_{k-1}}\}$ vérifie la k-cohérence si et seulement si toute instanciation de ses $k-1$ variables vérifient la k-cohérence.

Un CSP est k-cohérent si et seulement si tout sous-ensemble de V de cardinal $k-1$ est k-cohérent.

En d'autres termes, un CSP est k-cohérent si et seulement si pour toute instanciation cohérente de $k-1$ variables, il existe dans le domaine de toute variable non instanciée une valeur prolongeant cette instanciation en une instanciation cohérente de k variables [Freuder, 1978].

4.2.1.3 Un autre point de vue : la B-arc cohérence

Les domaines des variables peuvent être parfois représentés par uniquement leur borne inférieure et supérieure (soit un intervalle plutôt que par l'énumération des valeurs contenues dans le domaine). C'est le cas par exemple des dates de début des activités dans un problème d'ordonnancement. La propagation de contraintes sur de telles variables est souvent réalisée par *B-arc cohérence* (Arc-B-Consistency) [Lhomme, 1993] qui n'est autre que l'*arc cohérence* restreinte aux bornes des domaines.

Définition 6 (B-arc cohérence)
Une contrainte c vérifie la B-arc cohérence si et seulement si pour chaque variable $v_i \in var(c)$ de domaine $D_{v_i} = [a_i, b_i]$, il existe au moins une valeur dans le domaine des autres variables de $var(c)$ qui satisfait c lorsque v_i est instanciée à a_i et lorsque v_i est instanciée à b_i.

Un CSP vérifie la B-arc cohérence si et seulement si toutes ses contraintes vérifient la B-arc cohérence.

Exemple 5 (B-arc cohérence) :
Reprenons l'exemple 4. La contrainte $c_1 : t_1 + 3 \leq t_2$ n'est pas B-cohérente car si nous affectons à la variable t_1 de domaine $D_1 = [0, 7]$ la valeur $b_1 = 7$, il n'existe pas dans le domaine de la variable t_2 de domaine $D_2 = [2, 6]$ une valeur qui satisfait la contrainte c_1. La valeur $a_1 = 7$ sera donc éliminée du domaine D_1.
De même, la contrainte $c_2 : t_1 + 3 \leq t_3$ n'est pas B-cohérente car en affectant à la variable t_3 de domaine $D_3 = [1, 10]$ la valeur $a_3 = 1$, il n'existe pas dans le domaine de la variable t_1 une valeur qui satisfait la contrainte c_2. La valeur $a_3 = 1$ sera éliminée du domaine D_3.

4.2.1.4 Une généralisation : la k-B-cohérence

La *k-B-cohérence* est une extension de la *B-cohérence* à k variables.

Définition 7 (k-B-cohérence)
Une contrainte c vérifie la k-B-cohérence si et seulement si pour chaque variable $v_i \in var(c) = \{v_{i_1}, v_{i_2}, ..., v_{i_k}\}$ de domaine $D_{v_i} = [a_i, b_i]$, il existe des valeurs dans les domaines des autres variables de $var(c)$ qui satisfont c lorsque v_i est instanciée à la valeur a_i et lorsque v_i est instanciée à la valeur b_i.

Un CSP vérifie la k-B-cohérence si et seulement si toutes ses contraintes vérifient la k-B-cohérence.

4.2.2 Application récursive d'une cohérence locale : propagation

La suppression d'une valeur dans un domaine (en utilisant par exemple une des techniques de cohérence présentées ci-dessus), peut entraîner que des valeurs des domaines d'autres variables deviennent alors incohérentes. Il est alors nécessaire de réitérer l'analyse du problème et de continuer jusqu'à ce qu'aucune nouvelle information puisse être déduite. On appelle ce calcul de point fixe la *propagation de contraintes*.

Exemple 6 (Propagation de contraintes) :
Le tableau 4.2 représente les différentes étapes de la propagation des contraintes du CSP défini précédement en utilisant la B-arc cohérence.

Contrainte	Domaines	
$c_1 : t_1 + 3 \leq t_2$	$D_1 = [0,3]$	$D_2 = [3,6]$
$c_2 : t_1 + 3 \leq t_3$	$D_1 = [0,3]$	$D_3 = [3,10]$
$c_3 : t_2 + 2 \leq t_4$	$D_2 = [3,6]$	$D_4 = [5,13]$
$c_4 : t_3 + 4 \leq t_4$	$D_3 = [3,9]$	$D_4 = [7,13]$

TAB. 4.2 – *Propagation de contraintes dans l'exemple 4*

Lorsque la propagation s'achève, nous obtenons un CSP (V, D', C). Ce CSP est équivalent au CSP initial *(V, D, C)* (c'est à dire qu'il possède le même ensemble de solutions) et s'appelle la *fermeture par P-cohérence* où P est la propriété (la cohérence locale) considérée. Si un des domaines de D' est vide alors le CSP original n'a lui-même pas de solution et nous dirons qu'il est *sur-contraint*.

Exemple 7 (Fermeture par B-arc cohérence) :
La fermeture par B-arc cohérence du CSP de l'exemple 4 est le CSP défini par :
Les variables : $V = \{t_1, t_2, t_3, t_4\}$
Les domaines : $D' = \{D'_1 = [0, 3], D'_2 = [3, 6], D'_3 = [3, 9], D'_4 = [7, 13]\}$
Les contraintes : $C = \{c_1 : t_1+3 \leq t_2, c_2 : t_1+3 \leq t_3, c_3 : t_2+2 \leq t_4, c_4 : t_3 + 4 \leq t_4\}$.

Dans le cas des CSP binaires, *Mohr et Henderson* [1986] sont les premiers à proposer un algorithme, nommé *AC-4*, réalisant la fermeture par arc cohérence avec une complexité temporelle optimale en $O(md^2)$, où m est le nombre de contraintes et d la taille maximale des domaines, et une complexité spatiale de l'ordre de $O(md^2)$. Cet algorithme a été amélioré par *Bessière et Cordier* [1993] qui ont proposé un algorithme *AC-6* ayant la même complexité temporelle que *AC-4*, un meilleur comportement moyen et une complexité spatiale de l'ordre de *O(md)*.

Cooper [1989] a proposé un algorithme optimal pour calculer la fermeture k-cohérente. Sa complexité est de $O(n^k d^k)$, où n est le nombre de variables et d la taille maximale des domaines. Vu le coût élevé du calcul pour de grandes valeurs de k, seules sont considérées généralement de faibles valeurs de k : $k = 2$ (l'arc cohérence) et $k = 3$ (la cohérence de chemin).

4.3 CSP et optimisation

Les CSP classiques visent principalement à résoudre un problème d'existence. Dans un environnement d'aide à la décision, il est parfois nécessaire de chercher une solution qui optimise un critère donné. C'est le cas par exemple pour les problèmes d'ordonnancement, d'allocation de ressources, de planification. On peut alors introduire le formalisme COP (*Constraint*

optimisation problem : problème d'optimisation de contraintes).

4.3.1 Notion de problème d'optimisation de contraintes

En reprenant les définitions du paragraphe 4.1, notons S l'ensemble des solutions d'un CSP et considérons le cas où la fonction objectif consiste à minimiser un critère donné alors :

Définition 8 (Problème d'optimisation de contraintes)
Un problème d'optimisation de contraintes est défini par un quadruplet (V, D, C, Z), où V est un ensemble fini de variables, D est l'ensemble des domaines associés à ces variables, C est l'ensemble des contraintes qui portent sur ces variables et Z est une fonction de $D_{v_1} \times D_{v_2} \times ... \times D_{v_n}$ *vers* $\mathbb{R}$ *(le critère à optimiser).*

Si nous posons $Z_{min} = \min_{a \in S} Z(a)$*, alors une solution du* COP *est une affectation réalisable a tel que* $Z(a) = Z_{min}$.

> **Exemple 8 (Problème d'optimisation de contraintes) :**
> Reprenons l'exemple 4. Chercher une solution qui minimise la date de fin de l'ordonnancement (C_{max}) est équivalent à chercher une solution du COP (V, D, C, Z) qui minimise la fonction Z définie de $D_{t_1} \times D_{t_2} \times D_{t_3} \times D_{t_4}$ vers $\mathbb{R}$ et qui associe à chaque quadruplet $a = (a_1, a_2, a_3, a_4)$ la valeur $\max_{i \in \{1,..,4\}}\{a_i + p_i\}$.
> Nous avons : $C_{max} = \min_{a \in P}\{\max_{i \in \{1,..,4\}}\{a_i + p_i\}\}$.

4.3.2 Résolution d'un problème d'optimisation de contraintes

La résolution d'un COP peut être abordée de différentes manières :

- Toute solution d'un problème d'optimisation de contraintes *(V, D, C, Z)* satisfait $Z(s) \leq ub$, où ub est une borne supérieure du critère à minimiser. Ainsi, une solution d'un COP est aussi une solution du problème de satisfaction de contraintes *(V, D, C')*, où C' contient toutes les contraintes de C auxquelles on ajoute la contrainte $Z(x) \leq ub$. Une première approche consiste donc à résoudre successivement le CSP *(V, D, C')* en réduisant à chaque itération la valeur de ub (*cf.* Algorithme 1).

- Une deuxième approche consiste à procéder par dichotomie. On sélectionne une valeur e de l'intervalle *[lb, ub]*, où lb est une borne inférieure et ub une borne supérieure du critère à minimiser. Si le CSP *(V, D, C')*, où C' contient toutes les contraintes de C et la contrainte $Z(x) \leq e$ admet une solution alors la borne supérieure ub devient e. Sinon la borne inférieure lb devient *(e + 1)*. Ce processus est répété jusqu'à obtenir $lb = ub$ (cf. Algorithme 2).

Pour les deux approches précédentes, on peut varier la méthode de prise en compte de la contrainte sur la valeur recherchée de l'optimum. En effet, on peut, soit (tel que présenté) résoudre à chaque nouvelle étape le nouveau CSP; soit intégrer la nouvelle contrainte au système courant et partir de l'exploration courante pour rechercher une nouvelle solution. Il semble qu'aucune des deux approches ne soit significativement plus efficace que l'autre.

En tout cas, cette façon de procéder permet d'utiliser les algorithmes de résolution de CSP (algorithmes très souvent particulièrement travaillés) pour résoudre un COP.

Algorithme 1: Première méthode de résolution d'un COP.

Données : une borne supérieure ub.

Optimise(csp : (V, D, C), Z: **critère**, ub: **entier**)
début
  tant que *il existe une solution* **faire**
    $c := (Z(x) \leq ub)$
    Résoudre le csp $(V, D, C \cup \{c\})$
    $ub := ub - 1$
fin

Algorithme 2: Deuxième méthode de résolution d'un COP.

Données : une borne inférieure lb.
une borne supérieure ub.

Optimise(csp : (V, D, C), Z: **critère**, lb: **entier**, ub: **entier**)
début
  tant que $lb \neq ub$ **faire**
    choisir $e \in [lb, ub]$
    $c := (Z(x) \leq e)$
    si *le csp* $(V, D, C \cup \{c\})$ *admet une solution* **alors**
      $ub := e$
    sinon
      $lb := e + 1$
fin

4.4 CSP dynamiques

Le modèle CSP que nous avons présenté précédemment est un modèle statique : le problème est défini dès le début du processus de la résolution et reste invariant tout au long de son déroulement. Ce modèle ne permet pas de traiter les problèmes dynamiques où des événements imprévus peuvent intervenir à tout moment et modifier les données du problème. D'où la nécessité d'introduire la dynamicité dans le modèle CSP.

4.4.1 Définition

Définition 9 (CSP **dynamiques [Dechter et Dechter, 1988]**)
Un problème de satisfaction de contraintes dynamique (Dynamic Constraint Satisfaction Problem ou DCSP*) est une suite* $P_0, P_1, ...$ *de* CSP *classiques, tels que deux problèmes successifs* P_i *et* P_{i+1} *ne diffèrent que par l'ajout ou le retrait de contraintes.*

Si le CSP P_{i+1} est obtenu à partir du CSP P_i après le retrait d'une ou plusieurs contraintes, alors nous dirons que P_{i+1} est une *relaxation* de P_i. De même, si nous passons du CSP P_i au CSP P_{i+1} en ajoutant une ou plusieurs contraintes, alors nous dirons que P_{i+1} est une *restriction* de P_i.

Notons que la résolution d'un CSP statique peut être vue comme la résolution d'une série de problèmes. En effet, une instanciation de valeurs ou une extension d'une instanciation

peut être interprétée comme un ajout de contraintes et un retour arrière comme un retrait de contraintes.

Exemple 9 (Extension et restriction d'un CSP) :
Reprenons l'exemple 4 et notons par P_0 le CSP initial (V, D, C).
Supposons que nous avons un événement qui nous oblige à réaliser les activités sur une ressource de capacité $R_1 = 1$ sur un horizon de temps $H = 17$. Le CSP P_0 deviendra le CSP P_1 $(V, D, C_1 = C \cup \{c_5\})$, où c_5 est la contrainte de ressource : $\forall t \in [0, 17] \sum_{i \in V} w(i,t) \leq 1$ avec $w(i,t) = 1$ *si* $r_i \leq t < (f_i + p_i)$ *et* 0 *sinon*. P_1 sera donc une extension de P_0.
Supposons, maintenant que suite à un événement, la contrainte de précédence $2 \rightarrow 4$ n'est plus prise en compte. Le CSP P_1 deviendra le CSP P_2 $(V, D, C_2 = C_1 \setminus \{c_3\})$. P_2 sera donc une restriction de P_1.

4.4.2 Résolution

Comme un DCSP est défini de façon incrémentale, une approche souhaitable pour sa résolution sera d'éviter, à chaque relaxation ou restriction du problème, de recommencer la recherche à zéro. Pour cela, il paraît judicieux de mémoriser et réutiliser des résultats obtenus lors des recherches précédentes.

Dans la littérature, il existe trois types d'approches pour résoudre un DCSP : une approche qui consiste à mémoriser et réutiliser des solutions, une autre qui mémorise et réutilise des contraintes, et une dernière qui combine les deux premières et utilise la notion d'*explication* [Jussien, 2001]. Une étude exhaustive sur les deux premières approches a été présentée par *Verfaillie et Schiex* [1995].

4.4.2.1 Mémorisation et réutilisation de solutions

Cette approche repose sur l'idée suivante : si s_i est une solution du problème P_i, alors une solution du problème P_{i+1} (qui est proche du problème P_i) serait proche de s_i. Plusieurs méthodes basées sur cette idée ont été proposées. On peut citer par exemple :

- La méthode *heuristique de choix de valeur* [Van Hentenryck et Provost, 1991] qui consiste à affecter en priorité aux variables leurs valeurs dans la solution précédente;
- La méthode *réparation heuristique* [Minton *et al.*, 1992] qui consiste à choisir une variable d'une contrainte violée et à lui affecter une autre valeur de façon à minimiser le nombre de contraintes violées;
- La méthode *local changes* [Verfaillie et Schiex, 1994] qui s'appuie sur un mécanisme de retour arrière très particulier. Au cours de la recherche et lorsqu'une valeur est affectée à une variable v_i, le mécanisme ne désaffecte que les variables dont l'affectation est incompatible avec celle de v_i.

4.4.2.2 Mémorisation et réutilisation de contraintes

Cette seconde approche consiste à mémoriser des contraintes qui sont produites au cours des recherches précédentes et à les réutiliser dans la recherche courante. Ces contraintes

peuvent être produites soit par les mécanismes de filtrage par cohérence locale, soit par le mécanisme de recherche lui-même.

• **Maintien de l'arc cohérence:** Les algorithmes de filtrage par arc cohérence se comportent bien quand il s'agit de prendre en compte une restriction. En effet, si nous partons des domaines filtrés du problème P_i, les propagations que nous avons à réaliser sont uniquement dues à la contrainte ajoutée. Par contre, lorsqu'il s'agit de prendre en compte une relaxation, ces algorithmes ne peuvent pas déterminer directement lesquelles des valeurs des domaines éliminées précédemment doivent être rajoutées du fait de la relaxation. En effet, nous n'avons aucune information sur les raisons des retraits de ces valeurs.

DnAC-4 est le premier algorithme décrémental qui permet de calculer la fermeture arc-cohérence. Proposé par *Bessière* [1991a], cet algorithme n'est autre qu'une version dynamique de l'algorithme *AC-4*. Son principe est le suivant :

- Lors d'une restriction, à chaque valeur retirée d'un domaine est associée une justification. Cette justification est la première contrainte rencontrée sur laquelle la valeur retirée est sans support.

- Lors d'une relaxation, grâce aux justifications, à chacun des domaines sont ajoutées des valeurs qui permettent de rétablir l'arc-cohérence.

Par la suite d'autres algorithmes décrémentaux de maintien de l'arc-cohérence ont été proposés comme, par exemple, l'algorithme *DnAC-6* [Debruyne, 1995] qui est une adaptation de *AC-6*.

Plus généralement, *Debruyne et al.* [2003] ont présenté un schéma général des algorithmes permettant le retrait incrémental de contraintes. Ce schéma englobe tous les algorithmes de retrait incrémental existant. En se basant sur la notion d'*explication*, les auteurs ont introduit quelques conditions nécessaires pour assurer l'exactitude de tout autre nouvel algorithme de retrait incrémental.

• **Les *nogoods* :** Une autre approche basée sur la mémorisation de contraintes produites lors d'une recherche de type retour arrière utilise la notion de *nogood* : une affectation partielle des variables dont il a été prouvé qu'elle ne mène à aucune solution. À chaque *nogood* est associée une justification, qui n'est autre que le sous-ensemble de contraintes qui ont permis de construire ce *nogood*. Ceci permet de réutiliser les *nogoods* dont les justifications sont incluses dans l'ensemble des contraintes du problème courant.

4.4.2.3 Des *nogoods* aux *explications*

Verfaillie et Schiex [1994] ont proposé une approche combinant les deux approches précédentes : mémorisation et réutilisation de solutions et de contraintes. Ils utilisent un mécanisme permettant la réutilisation de solutions et des éliminations. La réutilisation des éliminations se fait grâce à la notion de *justification* d'élimination de valeurs. Une justification d'élimination d'une valeur est un ensemble de variables précédemment affectées et de contraintes du problème responsables de cette élimination.

Une généralisation de cette notion conduit à la notion d'*explication* [Jussien, 2001]. Une explication est définie comme un ensemble de contraintes expliquant une conséquence du solveur qui peut être un retrait de valeurs, une contradiction, ... Cette notion est à la base de nos travaux et sera présentée plus en détails dans le chapitre 5.

4.5 CSP sur-contraints

L'ajout incrémental de contraintes conduit dans certains cas à un CSP sur-contraint (*i.e.* pour lequel il n'existe pas de solution satisfaisant toutes les contraintes). Il faut alors chercher une solution ne respectant pas un ensemble minimal (au sens d'un critère déterminé) de contraintes. Plusieurs approches ont été développées dans ce sens. On peut distinguer :

- Des approches qui reposent sur un critère de cardinalité : l'objectif est de minimiser le nombre de contraintes violées. On peut ainsi citer les problèmes de type *Max-CSP* introduits par *Freuder et Wallace* [1992].

- Des approches reposant sur un critère de qualité : l'objectif est de satisfaire en priorité les contraintes les plus importantes. Nous citons par exemple le modèle *CSP valué* [Schiex et Verfaillie, 1995] où un poids est associé à chaque contrainte. Le but est alors de trouver une solution qui optimise un critère qui est fonction de ces poids.

- Des approches qui fournissent à l'utilisateur un ensemble de contraintes responsables de cet échec. L'utilisateur peut ensuite relaxer lui-même certaines de ces contraintes jusqu'à obtenir une solution satisfaisante. Nous pouvons citer l'approche proposée par *Jussien* [1997b] basée sur la notion d'*explication*.

4.6 CSP et problèmes d'ordonnancement

Comme nous l'avons vu dans l'exemple 4 (page 37), un problème d'ordonnancement peut être facilement modélisé en un CSP. Les techniques utilisées pour la résolution des CSP, en particulier les algorithmes de filtrages, sont facilement intégrables dans les techniques de résolution issues de la recherche opérationnelle. Plusieurs travaux ont été développés pour la résolution des problèmes d'ordonnancement en utilisant des techniques de résolution des CSP.

Ainsi, *Baptiste* [1998] a utilisé des techniques de la programmation par contraintes pour la résolution des problèmes d'ordonnancement à une machine, des problèmes de *Job-Shop* et des problèmes d'ordonnancement cumulatifs. *Laburthe* [1998] a utilisé ces techniques pour la résolution de nombreux problèmes combinatoires. Parmi eux, nous citons les problèmes d'ordonnancement disjonctifs, le problème du voyageur de commerce, les problèmes de tournées de véhicules et les problèmes d'ordonnancement cumulatifs. Nous pouvons citer aussi les travaux de *Guéret et al.* [2000] sur les problèmes d'*Open-Shop*.

Aussi, plusieurs techniques utilisées pour la résolution des problèmes de satisfaction de contraintes temporelles (Temporal Constraint Satisfaction Problems ou TCSP) ont été utile pour la résolution des problèmes d'ordonnancement. Les TCSP sont une classe particuliére des CSP : les variables représentent le temps et les contraintes représentent les ensembles des relations temporelles entre ces variables (pour plus de détailles voir [Schwalb et Vila, 1998] et [Huguet et Lopez, 1999]).

Tous ces travaux ont montré l'intérêt de l'utilisation des techniques de la programmation par contraintes pour la résolution des problèmes d'ordonnancement.

Un problème d'ordonnancement dynamique peut être interprété comme un problème de satisfaction de contraintes dynamique. En effet, un problème d'ordonnancement dynamique peut être considéré comme une série de problèmes qui différent les uns des autres par l'ajout ou le retrait d'événements ou de décisions.

Dans [El Sakkout et Wallace, 2000], les auteurs ont proposé une méthode hybride combinant programmation par contraintes et programmation linéaire permettant de produire des configurations minimales des ordonnancements en cas de perturbations. Un sous-ensemble de contraintes, généré grâce à un programme linéaire, est relaxé de manière à ce que ses solutions sont entiers afin de faciliter leur intégration dans une recherche basée sur la programmation par contraintes.

Les techniques de résolution des problèmes de satisfaction de contraintes dynamiques sont peu utilisées pour la résolution des problèmes d'ordonnancement dans le cadre dynamique. Ceci peut être dû à la difficulté rencontrée lors de l'application de ces techniques. En effet, cela nécessite d'avoir une bonne connaissance de ces techniques et de la façon de les appliquer. D'autre part, il existe peu de bibliothéques capable de gérer ces techniques.

4.7 Conclusion

L'objectif principal de ce chapitre est de présenter les différentes techniques existantes pour la résolution des *problèmes de satisfaction de contraintes* et les problématiques liées à l'*optimisation*, aux *problèmes dynamiques* et *sur-contraints*.

L'intégration des techniques issues de la programmation par contraintes pour la résolution des problèmes d'ordonnancement reste encore limité. En effet, la plupart des travaux n'utilisent que quelques techniques comme, le filtrage, le choix des variables à instanciées, le choix des valeurs et les techniques de retour arrière. Alors que d'autres techniques comme les *nogoods* ou les *explications* peuvent être très utiles pour la réduction de l'espace de recherche ou l'amélioration des stratégies de recherche.

Signalons que le but commun des *règles de dominance*, des *nogoods* et l'*arc cohérence* est de réduire le nombre de branches à explorer au cours de la recherche. Les *règles de dominance* établissent la dominance d'un nœud sur un autre, alors que les *nogoods* éliminent les branches dont nous sommes sûrs qu'ils ne conduiront pas à une solution. Une coopération entre ces deux différents concepts peut conduire à des résultats intéressants.

Plusieurs solveurs de contraintes ont été développés pour la résolution des problèmes combinatoires. La plupart de ces solveurs sont commerciaux : ils ne peuvent être ni modifiés ni étendus. Le solveur CHOCO, qui peut être aussi considéré comme une bibliothèque, a la particularité d'être extensible (son architecture peut être étendu à une large famille de contraintes, algorithmes de résolution ...), et d'offrir des outils de base (structures de données et algorithmes) nécessaire à tous système de contraintes. Parmis les extensions du solveur CHOCO, nous citons le solveur de contraintes PaLM permettant la gestion des explications. Le solveur (ou bibliothèque) PaLM sera présentée dans le chapitre 5.

Dans nos travaux, nous avons choisi d'utiliser la notion d'*explication* pour les raisons suivantes :

- Premièrement, car elle peut être utilisée pour la résolution des problèmes dynamiques et aussi pour des problèmes sur-contraints.
- Deuxièmement, l'utilisation des explications a donné de bons résultats pour la résolution de plusieurs problèmes combinatoires.
- Et enfin, nous avons à notre disposition une bibliothéque PaLM permettant de gérer les explications.

Chapitre 5

Programmation par contraintes avec explications

Sommaire

Introduction

Dans le chapitre 4, nous avons présenté les principales techniques rencontrées dans la littérature pour la résolution des CSP dynamiques et sur-contraints. Nous en avons retenu celle basée sur la notion d'explication.

Après avoir présenté les principaux mécanismes pour la résolution des CSP, nous introduisons, dans ce chapitre, la notion d'explication et présentons quelques applications des explications. Enfin, nous présentons la bibliothèque PaLM qui est l'environnement de programmation de notre système.

5.1 Mécanismes de résolution des CSP

Dans le chapitre précédent (*cf.* section 4.2 page 37), nous avons présenté quelques techniques (comme les cohérences locales et la notion de propagation) utilisées pour la résolution des CSP. Dans cette section, nous allons décrire plus en détail les procédures de résolution des CSP.

5.1.1 Recherche de solutions d'un CSP

La résolution d'un CSP repose bien souvent sur un algorithme basé sur le *retour arrière*. Dans celui-ci les variables sont instanciées séquentiellement. À chaque fois qu'une variable est instanciée, nous vérifions si le nouveau sous-ensemble de variables instanciées vérifie l'ensemble des contraintes qui les relient. Si une incohérence est détectée, un retour arrière est effectué sur la dernière variable instanciée. Une autre valeur du domaine de cette variable est alors choisie. Cette technique permet d'éliminer un sous-ensemble du produit cartésien des valeurs de toutes les variables à chaque fois qu'une instanciation partielle viole les contraintes du problème.

Malheureusement, cette méthode souffre de certains défauts connus sous le nom de *thrashing* [Gaschnig, 1979] : c'est à dire que le processus de recherche dans l'espace de solutions potentielles échoue souvent pour les mêmes raisons. En fait, l'algorithme effectue un parcours *aveugle* de l'espace de recherche sans prendre en compte des informations *évidentes* sur l'incohérence globale d'instanciations partielles et redécouvre de manière répétée les mêmes incohérences locales.

De nombreuses recherches ont été effectuées afin d'améliorer la prise en compte des causes d'incohérences pour le retour arrière. Citons :

- *les techniques de simplifications utilisées avant la recherche* comme par exemple la technique de cohérence locale (*cf.* section 4.2.1 page 38).

- *les techniques utilisées durant la recherche* : nous distinguons les techniques de type *look-back* qui consistent, en cas d'incohérence, à analyser l'instanciation partielle afin de déterminer les affectations responsables de l'échec. Différents algorithmes ont été proposés : *backmarking* [Gaschnig, 1977], *backjumping* [Gaschnig, 1979], *conflict-set* [Dechter, 1986], *graph-based backjumping* [Dechter, 1990], *dynamic backtracking* [Ginsberg, 1993], *conflict-directed backjumping* [Prosser, 1993] .... Et les techniques *look-ahead* qui sont chargées d'assurer la cohérence du problème pour l'instanciation courante. Citons par exemple : *forward checking, partial lookahead, full lookahead* [Haralick et Elliot, 1980], *maintien*

d'arc consistance ou mac [Sabin et Freuder, 1994]. Citons aussi l'algorithme *mac-dbt* [Jussien *et al.*, 2000] qui combine ces deux types de techniques.

– *les techniques utilisées pour guider la recherche* : ce sont des heuristiques produisant des ordres d'instanciations pour les variables et des ordres de choix de valeurs.

5.1.2 Utilisation d'un solveur de contraintes

Dans la pratique, il existe plusieurs manières d'aborder la résolution des CSP. Nous pouvons créer notre propre solveur ou utiliser un des solveurs ou bibliothèques existants. Nous pouvons alors :

• *Utiliser un langage de construction de solveurs* : l'utilisateur décrit (ou programme) dans un langage spécialisé le solveur de contraintes qu'il désire obtenir. Cette approche est illustrée par LAURE [Caseau, 1994] et CLAIRE [Caseau et Laburthe, 1996a]. L'utilisateur doit intégrer des mécanismes de bas niveau pour créer un solveur. Le langage lui fournit par exemple le moyen de stocker l'état courant des variables. Ainsi, l'utilisateur peut par exemple définir sa propre contrainte, sa propre stratégie de recherche (*cf.* l'exemple 10) ...

Exemple 10 (Stratégie de recherche) :
Dans le langage CLAIRE, une stratégie de recherche simple peut être définie comme suit :

```
[Branch(x : décision) : boolean
    try(
        world+(),
        if x true else (world-(), false)),
    catch contradiction (world-(), false)]
```

$world^+()$ créer un point de choix, et $world^-()$ crée un retour arrière. Le mécanisme **try** et **catch** permet de capturer toutes les exceptions qui peuvent se produire durant l'évaluation de la décision x tant qu'ils n'y a pas de contradiction. Lorsqu'une contradiction est détectée, un retour arrière est alors effectué.

Pour réaliser un bon solveur, il est nécessaire d'avoir une bonne connaissance des mécanismes de résolution de contraintes. Cependant il arrive régulièrement que les solveurs produits soient spécifiques à un problème donné pour être efficaces.

• *Utiliser une bibliothèque fournissant un ensemble de contraintes génériques ainsi que le mécanisme de résolution adapté* : l'utilisateur de telles bibliothèques doit traduire son problème dans le formalisme de contraintes offert par la bibliothèque. Cette traduction, selon le problème et la bibliothèque, peut être une tâche difficile. Les contraintes globales de CHIP, introduites par [Beldiceanu et Contejean, 1994, Aggoun et Baldiceanu, 1993], sont une illustration de telles bibliothèques. Les solveurs sont génériques et en général simples d'utilisation, cependant leur efficacité change suivant le domaine d'application.

• *Utiliser une bibliothèque extensible* : cette bibliothèque fournit les services d'une bibliothèque de contraintes, mais permet aussi à l'utilisateur de les spécialiser pour un domaine particulier. L'intérêt d'une telle approche est de permettre au programmeur une utilisation immédiate du solveur, tout en lui permettant de l'étendre ou de le spécialiser pour son utilisation. Citons par exemple la bibliothèque CHOCO [Laburthe, 2000] qui permet de traiter les contraintes unaires, binaires et linéaires, et la bibliothèque PALM [Jussien et Barichard, 2000]

permettant de traiter ces mêmes contraintes dans une version *expliquée*.

5.1.3 Réduction des domaines

La réduction des domaines dépend du type de domaines considérés (intervalles ou ensembles) et de la cohérence locale utilisée. Nous distinguons quatre types de réduction de domaines :

• Retirer une valeur val du domaine d'une variable x : la méthode permettant ce retrait est notée `removeValue(x,val)`.

• Mettre à jour la valeur de la borne inférieure du domaine de la variable x à la valeur val_inf : la méthode qui permet cette mise à jour est notée `updateInf(x,`val_inf`)`.

• Mettre à jour la valeur de la borne supérieure du domaine de la variable x à la valeur val_sup : la méthode permettant cette mise à jour est notée `updateSup(x,`val_sup`)`.

• Instancier la variable x à la valeur val : la méthode consiste à réduire le domaine de la variable x au singleton $\{val\}$. La méthode permettant cette instanciation est notée `instantiate(x,val)`.

Dans l'exemple 11, nous considérons un CSP dont les domaines sont définis par des intervalles et nous appliquons la *B-arc cohérence* (*cf.* section 4.2.1.3 page 39) comme cohérence locale. Les réductions de domaines sont alors assurées par les deux méthodes `updateInf` et `updateSup`.

> **Exemple 11 (Contrainte de précédence) :**
> Considérons un problème d'ordonnancement comportant trois activités ayant chacune une durée opératoire $p_i = 3$, une date de démarrage au plus tôt $r_i = 0$ et une date de démarrage au plus tard $f_i = 6$.
> Ces activités sont liées par les relations de précédence : $c_1 : t_1 + 3 \leq t_2$ et $c_2 : t_2 + 3 \leq t_3$.
> À cause de la contrainte c_1, la borne supérieure du domaine $D_1 = [0,6]$ de la variable t_1 est réduite à la valeur 3. Ce qui consiste à appliquer la méthode `updateSup(`t_1`,3)`.
> De même, à cause de la même contrainte c_1, la borne inférieure du domaine $D_2 = [0,6]$ de la variable t_2 va passée de la valeur 0 à la valeur 3. Nous appliquons alors la méthode `updateInf(`t_2`,3)`.

5.1.4 Mécanisme de propagation

Nous avons présenté précédemment les différentes méthodes existantes (`removeValue`, `updateInf`, `updateSup` et `instantiate`) pour réduire les domaines des variables. La réduction des domaines des variables dépend essentiellement de la contrainte qui relie ces variables et de l'état des domaines au moment de la propagation de cette contrainte. Ainsi, une contrainte peut être caractérisée par son comportement vis à vis des modifications des variables imposées par les environnements[1]. Des opérateurs de cohérence locale sont associés aux contraintes. Un tel opérateur peut être représenté par un couple (W_{in}, W_{out}), où W_{in}, $W_{out} \subseteq V$ (V est l'ensemble des variables du CSP). Dans la pratique, un opérateur permet de

1. Un environnement représente l'état des domaines des variables lors d'une propagation. Il peut être considéré comme un sous-ensemble du produit cartésien des domaines des variables du CSP.

retirer, des environnements de W_{out}, quelques valeurs qui sont incohérentes et qui respectent les environnements de W_{in}.

La propagation est gérée grâce à une *file de propagation* contenant des événements ou des opérateurs à réveiller. Plus précisement, à partir d'un environnement initial du problème, un opérateur de la cohérence locale est séléctionné dans la file et appliqué à l'environnement conduisant ainsi à un nouvel environnement. Ce qui permet d'ajouter de nouveaux événements ou opérateurs à la file de propagation.

Ainsi, dans CHOCO, le mécanisme de propagation est basé sur la propagation des événements : `removeValue(x,val)`, `updateInf(x,`val_inf`)`, `updateSup(x,`val_sup`)` et `instantiate(x,val)`. À chaque fois que le domaine d'une variable est modifié, le domaine de la variable est aussitôt mis à jour et toutes les contraintes connectées à cette variable réagissent à cet événement. La réaction à un tel événement se fait par envoi de messages qui sont des appels aux méthodes :

• `awakeOnRem(C:Constraint, i:integer, val:integer)` qui consiste à propager la contrainte C suite à un événement `removeValue(x,val)` si x est la $i^{éme}$ variable de la contrainte C.

• `awakeOnInf(C:Constraint, i:integer)` qui consiste à propager la contrainte C suite à un événement `updateInf(x,`val_inf`)` lorsque x est la $i^{éme}$ variable de la contrainte C.

• `awakeOnSup(C:Constraint, i:integer)` qui consiste à propager la contrainte C suite à un événement `updateSup(x,`val_sup`)` si x est la $i^{éme}$ variable de la contrainte C.

• `awakeOnInst(C:Constraint, i:integer)` qui consiste à propager la contrainte C suite à un événement `instantiate(x,val)` si x est la $i^{éme}$ variable de la contrainte C.

Ces méthodes peuvent être considérées comme des opérateurs de cohérence locale. Nous présentons, dans l'exemple 12, l'opérateur `awakeOnSup` lié à la contrainte binaire $C : x \geq y+c$.

La figure 5.1 représente l'architecture de CHOCO en termes d'événements. Dans cet exemple, nous avons 6 variables et 4 contraintes. La contrainte C_2 porte sur les trois variables x_2, x_6 et x_4. Lorsque cette contrainte reçoit un message indiquant que le domaine de la variable x_6 a été réduit, les domaines des deux variables x_2 et x_4 seront aussitôt réduits. Ces réductions permettent de renvoyer à leurs tours d'autres messages (la variable x_2 envoie un message à la contrainte C_4).

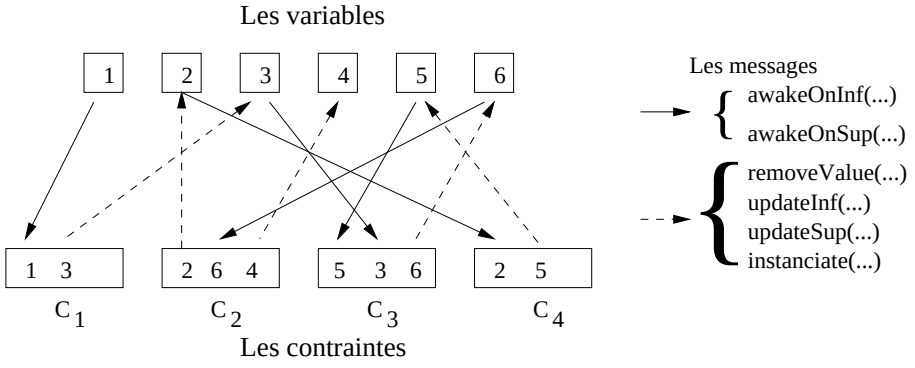


FIG. 5.1 – *Architecture de* CHOCO *en termes d'agents et événements.*

Exemple 12 (Contrainte $C : x \geq y + c$) :
Représentée par la classe `GreaterOrEqualxyc`, la contrainte $x \geq y+c$ constitue une des contraintes de base de CHOCO. La réaction pour une mise à jour de la borne supérieure de l'une des variables de cette contrainte (à la variable x est associée l'indice `idx = 1` et à y l'indice `idx = 2`) est comme suit :
Si la borne supérieure $x.sup$ de x est modifiée alors la borne supérieure de y doit être réduite à une nouvelle valeur. Ceci peut être coder de la façon suivante :

```
[awakeOnSup(C:GreaterOrEqualxyc,idx:integer)
-> if (idx = 1)
     updateSup(y, x.sup - c)]
```

Notons que la méthode `updateSup` ne modifie la valeur de y que si sa borne supérieure a réellement changée.
Dans cet exemple, l'opérateur `awakeOnSup` est représenté par le couple (W_{in}, W_{out}), où $W_{in} = \{x\}$ et $W_{out} = \{y\}$.

5.1.5 Énumération

La propagation de contraintes n'est pas toujours suffisante pour trouver une solution. Une phase d'énumération est alors nécessaire. La phase d'énumération peut être considérée comme une série d'ajouts et de retraits de contraintes. Ces contraintes seront appelées **contraintes d'énumération**. Elles peuvent être des affectations de valeurs ou des contraintes de décision.

Au cours de la résolution, le système de contraintes évolue en ajoutant ou retirant des contraintes d'énumération. Á chaque état de la résolution du CSP correspond deux ensembles de contraintes : l'ensemble des contraintes originales du problème et l'ensemble des contraintes d'énumération. Le second ensemble représente le chemin courant dans l'arbre de recherche qui a été exploré.

5.2 Explications pour la propagation de contraintes

Considérons un CSP (V, D, C), où V est un ensemble fini de variables, D est l'ensemble des domaines associés à ces variables et C est l'ensemble des contraintes qui portent sur ces variables. Et notons par c_i les contraintes d'énumération introduites au cours de la recherche.

Une explication peut être définie comme un ensemble de contraintes justifiant une action du solveur (retrait d'une valeur d'un domaine, réduction d'un domaine, contradiction, ...). Deux types d'explications sont alors définies : les explications liées au retrait ou à la réduction d'un domaine et les explications liées à un échec.

5.2.1 Explication de retrait

Une *explication de retrait* est définie comme un ensemble de contraintes justifiant la réduction d'un domaine. Ainsi, si nous supposons qu'un domaine d'une variable a été réduit suite à l'ajout des contraintes d'énumération c_i, $i \in [1..k]$, l'explication de ce retrait sera donc :

$$\texttt{expl}(v_j \neq a_j) = C \wedge \bigwedge_{i \in [1..k]} (c_i)$$

Notons qu'il existe, en général, plusieurs explications différentes pour un retrait donné. Conserver toutes ces explications ne serait pas possible à cause de la complexité exponentielle en espace [Piechowiak et Rodriguez, 2000]. Une solution consistera à ne considérer que celles qui sont *pertinentes*[2] dans l'état actuel de la recherche. De cette façon, la complexité spatiale de l'enregistrement reste polynomiale [Jussien, 1997b]. L'approche choisie consiste à ne conserver qu'une seule explication de retrait à la fois.

5.2.2 Explication de contradiction

Une *explication de contradiction*, appelée aussi *nogood*, est définie comme un sous-ensemble incohérent du système courant de contraintes.

Comme une contradiction n'est détectée que lorsque le domaine de l'une des variables de V devient vide, les explications de contradiction peuvent être alors calculées comme suit :

$$\bigwedge_{a_j \in D_{v_j}} \texttt{expl}(v_j \neq a_j)$$

Les explications de contradiction (valable aussi pour le retrait) les plus intéressantes sont celles qui sont minimales au sens de l'inclusion. Ainsi, une explication $\texttt{E}$ est dite minimale si et seulement si pour tout sous-ensemble strict $\texttt{E}'$ de $\texttt{E}$, le CSP $(V, D, \texttt{E}')$ est cohérent (*i.e.* admet une solution).

5.3 Calcul des explications

Pour produire des explications, il faut connaître, pour chaque contrainte, les raisons du retrait des valeurs des domaines de ces variables. De plus, il faut utiliser ces connaissances pour fournir des explications relativement précises pour qu'elles soient intéressantes.

5.3.1 Contraintes de base

Fournir des explications pour des contraintes de base est facile. L'exemple suivant nous donne une idée sur le calcul de ces explications.

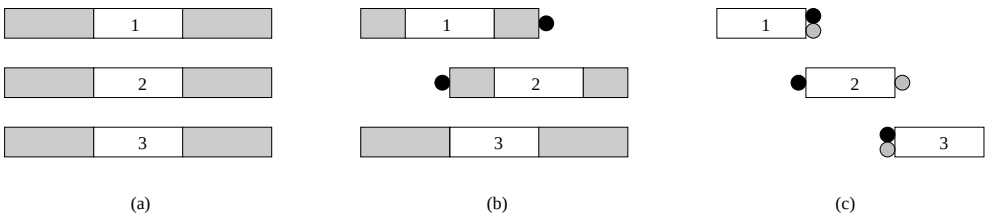


FIG. 5.2 – *Dans cette figure, les points noirs représentent les bornes réduites à cause de la contrainte c_1 et les points gris indiquent les bornes réduites à cause de la contrainte c_2.*

2. Une explication de retrait est dite pertinente si toutes les contraintes la constituant sont encore valides dans l'état courant du système.

Exemple 13 (Calcul des explications) :
Considérons le problème d'ordonnancement défini dans l'exemple 11 (page 50). Une représentation de ce problème est donnée dans la figure 5.2 (a).
Comme nous l'avons vu précédemment, la borne supérieure du domaine D_1 associé à la variable t_1 a été réduite à la valeur 3. De même, la borne inférieure du domaine D_2 associé à la variable t_2 a été réduite à la valeur 3 (*Fig.* 5.2 (b)). Nous avons donc, $\texttt{theSup}(t_1) = \{c_1\}$ et $\texttt{theInf}(t_2) = \{c_1\}$.
La propagation de la contrainte c_2 entraîne la réduction de la borne inférieure du domaine D_3 de la variable t_3, D_3 devient donc $\{6\}$. Cette réduction est causée par la contrainte c_2 et les contraintes responsables de la réduction de la borne inférieure du domaine de t_2 (ici c'est c_1). Nous avons $\texttt{theInf}(t_3) = \{c_1, c_2\}$.
De même, les domaines D_1 et D_2 sont respectivement réduits à $\{0\}$ et à $\{3\}$. Nous avons $\texttt{theInf}(t_2) = \{c_1\}$, $\texttt{theSup}(t_2) = \{c_2\}$ et $\texttt{theSup}(t_1) = \{c_1, c_2\}$ (*Fig.* 5.2 (c)).

5.3.2 Contraintes globales

L'implémentation d'une contrainte globale est essentiellement basée sur l'algorithme de filtrage qui lui est associé. Ceci rend la tâche du calcul d'explications précises [3] très difficile. En effet, nous devons donner une explication pour chacune des décisions prises lors du filtrage. Ces décisions peuvent être soit des retraits de valeurs, des réductions des bornes ou des incohérences.

Dans [Rochart et Jussien, 2003], les auteurs se sont intéressés à la contrainte globale *stretch* [Pesant, 2001]. En se basant sur l'algorithme de filtrage proposé par *Pesant* pour cette contrainte, les auteurs ont montré comment étendre une telle contrainte pour pouvoir l'utiliser dans un contexte expliqué. Des explications précises pour chacune des décisions prises lors du filtrage sont alors produites. Dans [Rochart *et al.*, 2003], les auteurs ont présenté, à travers des exemples, quelques démarches permettant de produire des explications précises pour des algorithmes de filtrages. Ils se sont intéressés aux contraintes globales *all-different*, *stretch* et *flow*.

Dans notre système, nous aurons besoin de traiter les contraintes de ressources. Ces contraintes seront modélisées comme des contraintes globales. Le calcul des explications associées à ces contraintes sera présenté ultérieurement.

5.4 Quelques applications des explications

Cette section présente quelques utilisations possibles des explications.

3. Nous pouvons donner comme explication (grossière) l'état courant des domaines des variables concernées. Mais, bien évidemment, dans le cas où la contrainte prend en compte toutes les variables du problème, l'explication ainsi fournie n'est pas vraiment utilisable.

5.4.1 Interaction entre l'utilisateur et le solveur

L'utilisation d'un solveur de contraintes basé sur les explications permet d'aider l'utilisateur en lui fournissant des réponses aux questions posées lors de la recherche d'une solution comme par exemple :

- *Pourquoi ce problème n'admet pas de solution ?*
  Un problème d'admet pas de solution lorsque le domaine d'une de ses variables v devient vide suite au retrait de toutes les valeurs de ce domaine. Produire une explication de cette situation aide à comprendre les raisons de cet échec en déterminant un sous-ensemble de contraintes responsables.

- *Pourquoi la variable v ne peut pas prendre la valeur a ?*
  Pour répondre à cette question, il suffit de consulter l'explication associée au retrait $(v \neq a)$ (*i.e.* $expl(v \neq a)$).

- *Que se passerait-il si nous réintroduisions une contrainte c ?*
  Dans cette situation, la contrainte c a été retirée. Les explications enregistrées contenant la contrainte c vont permettre d'avoir un apérçu des conséquences du retour de la contrainte c.

5.4.2 Traitement incrémental des problèmes dynamiques

5.4.2.1 Retrait décrémental de contraintes

Un système d'explications peut être utilisé pour la gestion des problèmes dynamiques. L'ajout incrémental de contraintes dans les systèmes de programmation par contraintes est bien connu. Par contre, le retrait décrémental est plus délicat.

Le retrait décrémental d'une contrainte peut être accompli de la façon suivante (*Bessière* [1991b], *Jussien* [2001], *Debruyne et al.* [2003]) :

• *Déconnexion :* La première étape consiste à retirer la contrainte c du réseau de contraintes représentant le problème courant. c doit être entièrement déconnectée et ne doit pas être propagée dans le futur. Déconnecter une contrainte c correspond à retirer tous ses opérateurs antérieurs de l'ensemble courant des opérateurs actifs.

• *Rétablissement :* La seconde étape consiste à défaire les effets passés (directs ou indirects) de la contrainte. Pour cela, il suffit de considérer toutes les explications contenant cette contrainte, invalider les événements associés (retraits de valeurs) à ces explications et rétablir ces valeurs dans leurs domaines. Cette étape conduit à l'élargissement de l'environnement.

• *Repropagation et dépropagation (ou contrôle) :* Quelques valeurs rétablies précédemment peuvent être retirées en appliquant d'autres opérateurs actifs (*i.e.* opérateurs associés aux contraintes non retirées). Ces réductions de l'environnement doivent être effectuées et propagées afin de rétablir un état cohérent.

À la fin de ce processus, le système obtenu est dans un état cohérent. Il correspond exactement à l'état qui aurait été obtenu si la contrainte retirée c n'avait pas été introduite dans le système.

Comme nous nous situons dans un environnement dynamique, cette procédure est intéressante dans le cas où nous voulons retirer décrémentalement des contraintes du système.

5.4.2.2 Résolution de problèmes sur-contraints

Pour résoudre un problème sur-contraint, il faut un outil capable de fournir une explication pour une contradiction et de retirer de manière décrémentale une contrainte dans un système de contraintes. Cet outil doit donc identifier la/les contrainte(s) à relaxer grâce aux explications. L'utilisateur peut alors relaxer la/les contrainte(s) selon son choix. Ce processus est réitéré jusqu'à obtenir une solution satisfaisante.

Nous nous sommes intéressés à ce processus, car il permet de créer une interaction entre l'utilisateur et le solveur, et aussi il fournit une aide à la résolution des problèmes sur-contraints.

5.4.3 Amélioration de l'efficacité des algorithmes de recherche

La notion d'explication peut être aussi utilisée pendant la résolution. En effet, la notion de retour arrière, qui n'est autre qu'un retrait de la contrainte précédente, peut être remplacée par la notion de réparation. La réparation consiste à trouver une explication d'une contradiction et à *inverser* une de ces contraintes.

Un exemple de l'utilisation des explications dans un algorithme de résolution est l'algorithme *mac-dbt* [Jussien *et al.*, 2000]. L'idée consiste à utiliser le maintien de l'arc-cohérence (*mac*) au sein de *dynamic backtracking* (*dbt*) [Ginsberg, 1993]. Cet algorithme a pour principal objectif d'expliquer non seulement chacune des contradictions comme *dynamic backtracking* mais aussi d'expliquer chacune des réductions de domaines effectuée lors de la propagation.

Le comportement général de *mac-dbt* est décrit dans l'algorithme 3. Tant qu'une solution n'est pas trouvée, choisir une variable et une valeur de son domaine, affecter la valeur à la variable (ligne 1), propager cette nouvelle information (ligne 2). Si une contradiction se produit durant ce processus, utiliser une procédure spéciale pour traiter la contradiction (ligne 3 et l'algorithme 4) qui remplace le retour arrière classique.

Dans l'algorithme 4, une explication de cet échec est calculée (ligne 1), et, si c'est possible, une contrainte d'affectation à défaire est selectionnée (ligne 2). La contrainte choisie est aussitôt retirée du système (ligne 3), sa négation est alors ajoutée et cette nouvelle information est propagée (ligne 4). Les contradictions pouvant apparaître sont traitées récursivement (ligne 5).

Dans cet algorithme, les décisions prises en chaque nœud de l'arbre de recherche sont des affectations de valeurs. Comme, dans notre système, ces décisions seront représentées par des contraintes, cet algorithme va donc être généralisé.

5.5 Le système PaLM

PaLM (*Propagation and Learning with Move*) est un système de programmation par contraintes basé sur les explications. Il est codé en langage CLAIRE [4] et utilise la bibliothèque CHOCO [5].

4. CLAIRE est un langage de programmation conçu pour exprimer élégament des algorithmes discrets complexes [Caseau et Laburthe, 1996a].

5. CHOCO est une bibliothèque, codée en CLAIRE, offrant des outils de base pour la programmation par contraintes : gestion des domaines, propagation de contraintes, procédures de recherche globales et locales [Laburthe, 2000].

Algorithme 3: L'algorithme mac-dbt.

```
Données : un problème pb
Résultat : booléen = vrai si une solution existe, faux sinon

Solve(pb: problème)
début
    unassignedVars ← pb.vars,
    try
        tant que not(empty(unassignedVars)) faire
            let
                idx ← nextVarToAssign(pb), // choix d'une variable
                v ← unassignedVars[idx],
                a ← selectValToAssign(pb, v)  // choix d'une valeur
            in
                try
                    unassignedVars :delete v,
1                   post(pb, v == a),  // instanciation
2                   propagate(pb)
                catch LabelingContradiction // un domaine vide a été trouvé
3                       handleContradiction(pb) // traitement de la contradiction
        true
    catch ProblemContradiction
            faux // pas de solution
fin
```

Le système PaLM fournit des outils qui permettent de gérer des explications.

- PaLM peut être considéré comme un solveur de contraintes **classique** : il peut être utilisé pour résoudre des problèmes comme si nous utilisons la bibliothèque choco.
- PaLM est un solveur de contraintes **dynamique** : il traite dynamiquement l'ajout et le retrait de contraintes avant, durant ou après la résolution.
- PaLM est un solveur de contraintes **basé sur les explications** : il fournit des algorithmes de recherche spécifiques (*cf.* section 5.4.3).

5.5.1 Enregistrement et consultation des explications

Le système PaLM fournit des outils qui permettent de calculer, mémoriser et consulter les explications pour chaque modification des domaines pour un problème donné.

Les explications sont enregistrées comme suit :

- pour les variables représentées par des bornes inférieures et supérieures, deux piles d'explications sont stockées : une pile pour chacune des bornes.
- pour les variables représentées par un ensemble de valeurs possibles, une explication est associée à chaque valeur.

Algorithme 4: Traitement de la contradiction.

```
handleContradiction(pb: problème)
début
    let
1       e ← theDomaine(getFailingVariable(pb))  // explication du conflit
    in
        si e empty alors
            raiseProblemContradiction()
        sinon
            let
2               ct ← selectConstraint(e)  // selection de la contrainte à retirer
            in
                si ct existe alors
                    unassignedVars :add ct.v1,
                    try
3                       remove(ct),  // retirer la contrainte
                        e :delete ct,
                        post(pb, opposite(ct), e),  // ajouter la négation de la contrainte
4                       propagate(pb)  // achever la consistence
                    catch LabelingContradiction
5                       handleContradiction(pb)  // traitement récursif
                sinon
                    raiseProblemContradiction()
fin
```

Le système PaLM fournit aussi une méthode générale permettant de construire les explications : `becauseOf`. Cette méthode permet de joindre une explication à une liste d'explications associées à l'information relative à l'événement. Les informations ou paramètres acceptés par cette méthode sont :

- `theSup(x)` (*resp.* `theInf(x)`) pour expliquer la borne supérieure (*resp.* inférieure) de la variable x. Si le domaine de la variable est représenté par ses deux bornes, cela revient à enregistrer la dernière entrée dans la pile associée. Par contre, si le domaine de la variable est représenté pas un ensemble de valeurs possibles, l'union de l'explication des valeurs inférieures (*resp.* supérieure) à la borne courante est retournée.

- `theDom(x)` pour expliquer le domaine courant de la variable x. Cela revient à joindre les explications de toutes les valeurs retirées du domaine de x.

- `theHole(x, v)` pour expliquer le retrait de la valeur v du domaine de la variable x.

- `theConstraint(c)` pour ajouter une contrainte donnée à l'explication calculée.

5.5.2 Intégration des explications

Comme les explications doivent être calculées et enregistrées lorsque des événements sont générés, l'ajout de l'information dans les méthodes `updateInf`, `updateSup` et `removeVal` est alors nécessaire.

L'exemple 14, donne un aperçu sur la façon de calculer les explications pour une contrainte de base.

Exemple 14 (Modification du solveur) :
Considérons la contrainte binaire $x \geq y + c$ (`GreaterOrEqualxyc`) présentée dans l'exemple 12.
Il est facile de modifier le code de la méthode `awakeOnSup`. En effet, toutes les informations sont disponibles dans cette méthode. Ainsi, la modification de la borne supérieure de la variable y est due à :

- l'utilisation de la contrainte elle même (elle doit être ajoutée à l'explication calculée).
- la modification précédente de la borne supérieure de la variable x qui peut être récupérée grâce au paramètre `theSup(x)`.

Le code source de la méthode `awakeOnSup` est alors modifié comme suit :

```
[awakeOnSup(C:GreaterOrEqualxyc,idx:integer)
-> if (idx = 1)
     updateSup(y, x.sup - c, becauseOf(theConstraint(C),
theSup(x)))]
```

5.6 Conclusion

Après avoir définie la notion d'explication, nous avons présenté, dans ce chapitre, quelques techniques permettant de les calculer. Nous avons aussi présenté quelques applications des explications et la bibliothéque PaLM qui permet de gérer les explications. Cette bibliothéque sera l'environnement de programmation de notre système.

Nous avons à notre disposition un système capable de gérer les explications et une stratégie de recherche (*mac-dbt*) adaptable à la résolution des problèmes dynamiques.

Pour concevoir notre système, nous aurons besoin dans un premier temps, de modéliser notre problème d'ordonnancement (RCPSP et ses extensions) en un problème de satisfaction de contraintes. Pour cela, les contraintes temporelles et de ressources de notre problème doivent être modélisées en contraintes simples à manupiler et les explications associées doivent être faciles à calculer. Ensuite, nous deverons avoir un schéma de séparation simple, efficace et adaptable à la stratégie de recherche *mac-dbt*.

Chapitre 6

Bilan

Nous venons d'une part de faire un état de l'art des différentes approches développées pour résoudre les problèmes d'ordonnancement de type RCPSP statiques ainsi que les problèmes d'ordonnancement dynamiques. D'autre part, nous avons passé en revue les techniques utilisées pour résoudre des CSP statiques, sur-contraints ou dynamiques.

Nous avons présenté dans le chapitre 2 les différentes approches qui ont été développées pour la résolution du RCPSP dans le cadre statique. Ainsi, nous avons vu que de nombreux travaux portent sur ces problèmes : de nombreuses bornes inférieures constructives ou destructives, plusieurs bornes supérieures (généralement basées sur des règles de priorité), des schémas de séparation très variés ainsi que diverses règles de déduction (généralement des extensions de règles conçues pour les problèmes d'ateliers). Cette étude nous a permis de détecter des approches performantes pour résoudre le cas statique qui pourront être réutilisées pour résoudre le cas dynamique. En particulier, on peut retenir que le schéma de séparation proposé par *Brucker et al.* [1998] est l'un des plus efficaces, et noter que les contraintes de ressources sont généralement vérifiées en utilisant des règles de déduction.

On notera de plus que le problème RCPSP classique étant déjà un problème complexe, peu de travaux concernent les extensions de ce problème (avec contraintes de précédence généralisées, variation des capacités des ressources, etc).

Le chapitre 3 concerne la résolution des problèmes d'ordonnancement dynamiques. Les méthodes proposées sont de trois types : proactives, réactives, ou proactives-réactives. Le nombre de travaux proposés dans la littérature est faible et les méthodes proposées ne sont pas satisfaisantes. En effet, dans ces méthodes les événements ou les décisions pris en considération sont très limités, et les travaux portent principalement sur des problèmes restreints à une machine. De plus, aucune des méthodes de la littérature n'assure l'optimalité. Nous n'avons en particulier pas trouvé dans la littérature de méthodes de type réactives optimales.

Une remarque importante suite à l'étude des techniques proposées que ce soit dans le cas statique ou dynamique, est qu'aucune d'elles ne permet de résoudre les problèmes qui n'ont pas de solution.

Dans une deuxième partie de l'état de l'art, nous nous sommes intéressés aux CSP (ou problèmes de satisfaction de contraintes) (*cf.* chapitre 4). Nous avons défini le formalisme CSP et présenté les techniques de résolution des CSP dans le cadre statique, sur-contraint, puis dynamique. Nous avons ainsi présenté les techniques de résolution des CSP et plus particulièrement la notion de filtrage, ainsi que les différentes techniques de résolution des CSP sur-contraints et dynamiques, en nous attardant plus particulièrement sur l'une de ces techniques : la notion d'explication. L'intérêt de cette approche est qu'elle permet de gérer aussi bien les problèmes dynamiques que les problèmes sur-contraints. Cette dernière a de plus déjà montré son intérêt pour la résolution de problèmes d'optimisation combinatoire, et en particulier pour les problèmes d'ordonnancement statiques.

Nous avons aussi dans ce chapitre montré comment un problème d'ordonnancement dynamique peut être modélisé comme un CSP dynamique.

Étant donné le manque de méthodes satisfaisantes pour la résolution de problèmes d'ordonnancement dynamiques, l'objectif de cette thèse est de concevoir un système capable de résoudre de tels problèmes. Ce système pourra traiter des **problèmes d'ordonnancement complexes**, sera **capable de réagir à plusieurs types d'événements**, de **répondre en un temps raisonnable** et de **fournir des ordonnancements successifs optimaux qui ne diffèrent pas trop les uns des autres**.

Nous allons nous intéresser, dans un premier temps, à un problème d'ordonnancement très général : le RCPSP, car il contient en cas particuliers un grand nombre de problèmes (les

problèmes à une machine, les problèmes à machines parallèles, les problèmes d'ateliers ...). Et dans un second temps, à quelques extensions du RCPSP.

Notre démarche sera la suivante : nous allons modéliser notre problème d'ordonnancement en un problème de satisfaction de contraintes. Nous devrons ensuite transformer les contraintes temporelles généralisées et les contraintes de ressources en des contraintes simples à manipuler.

Pour avoir un système dynamique, nous aurons besoin de modéliser les différents événements sous forme de contraintes, et d'un système qui gère ces contraintes et qui calcule les explications associées à leur propagation.

Notre système sera plus avantageux que les approches proposées actuellement dans la littérature pour plusieurs raisons :

- il pourra traiter plusieurs types d'événements ou de décisions : il suffit de modéliser ces événements sous forme de contraintes et de calculer les explications associées;
- le temps de calcul sera *a priori* peu coûteux : la solution ne sera pas reconstruite à partir de zéro à chaque arrivée d'un nouvel événement, mais se basera sur la solution courante;
- les solutions successives ne différeront *a priori* pas trop les unes des autres: nous ne réalisons que des modifications utiles sur la solution courante;
- dans le cas où le problème n'a pas de solution, le système fournira les contraintes à retirer pour obtenir une solution.

Deuxième partie

Résolution du RCPSP dynamique

Introduction

Cette partie traite la résolution du problème d'ordonnancement à contraintes de ressources (RCPSP) dans le cadre dynamique.

Notre démarche consiste à modéliser le RCPSP en un CSP. Et de lui appliquer les techniques existantes pour la résolution des CSP dynamiques. Ces techniques sont essentellement basées sur les explications.

Cette partie est organisée comme suit :

• Le premier chapitre présente notre environnement de résolution. Nous présentons un nouveau schéma de séparation inspiré de celui de *Brucker et al.*. Nous définissons la notion de *distance* qui nous permettera de modéliser les contraintes temporelles imposées par le problème et les contraintes utilisées dans le schéma de séparation en des contraintes simples. Et enfin, nous présentons l'algorithme utilisé pour la recherche d'une solution optimale.

• Le second chapitre présente le mécanisme de propagation des contraintes temporelles (définies dans le premier chapitre) et le calcul des explications associées.

• Le troisième chapitre traite les contraintes de ressources. Nous présentons deux types de contraintes : une basée sur la notion de *parties obligatoires* et de l'*histogramme*, et l'autre sur la notion d'*intervalles de tâches*. Nous présentons, pour chacune de ces contraintes, le mécanisme permettant de maintenir les structures des données, le mécanisme de propagation basé sur des règles de déductions, et enfin les explications associées. Nous évoquons aussi dans ce chapitre les différents événements pris en compte puis nous traitons, à travers un exemple, le cas sur-contraint.

• Le dernier chapitre est consacré aux expérimentations. Nous présentons dans un premier temps les résultats obtenus dans le cadre statique. Puis ceux obtenus pour le RCPSP dynamique.

Chapitre 7

Un environnement de résolution dynamique

Sommaire

Introduction

Dans ce chapitre, nous décrivons notre environnement de résolution du RCPSP dans le cadre dynamique. Nous présentons notre schéma de séparation inspiré de celui développé par *Brucker et al.* [1998], la notion de distance qui nous a permis de modéliser les contraintes de décisions (prises au cours de la recherche) ainsi que leur négations en des contraintes simples, et enfin, l'algorithme utilisé pour la résolution optimale du RCPSP.

7.1 Schéma de séparation

La stratégie de recherche présentée dans le chapitre 5 (*cf.* section 5.4.3 page 56) est la plus adaptée à la résolution des problèmes dynamiques. Dans cette stratégie, les contraintes d'énumération prises au cours de la recherche sont des contraintes d'affectation. Nous avons généralisé cette stratégie de recherche pour le cas où les contraintes d'énumération sont des contraintes de décision.

Le principe de notre stratégie de recherche est le suivant : une décision prise en un nœud de l'arbre de recherche est traduite par l'ajout d'une contrainte. Lorsqu'une contradiction apparaît au cours de la recherche, une contrainte de l'explication de cette contradiction (la plus récente) est retirée du système et sa négation est ajoutée. Pour assurer la complétude de la recherche, la contrainte retirée est la plus récente parmi les contraintes de l'explication de contradiction [Ginsberg, 1993, Jussien *et al.*, 2000].

Afin d'intégrer cette stratégie dans notre système, il est nécessaire d'avoir un schéma de séparation dans lequel les décisions prises en chaque nœud, ainsi que leurs négations soient modélisables par des contraintes simples. Les explications associées à ces contraintes doivent être facilement calculables. En vue d'étendre notre système à des problèmes d'ordonnancement dynamiques plus généraux, il est préférable que ce schéma de séparation soit aussi utilisable pour la résolution des extensions du RCPSP.

Parmi les schémas de séparation que nous avons présentés dans le chapitre 2, nous avons retenu celui développé par *Brucker et al.* (*cf.* 2.4.4 *page* 14). Une des raisons de ce choix est qu'il a donné de bons résultats pour la résolution exacte du RCPSP [Brucker *et al.*, 1998].

L'application de ce schéma de séparation à notre démarche est assez délicate. En effet, lorsqu'une feuille de l'arbre de recherche est atteinte, des disjonctions doivent être orientées afin d'obtenir un ordonnancement ne contenant aucune relation de disjonction. Puis, dans le cas où l'ordonnancement obtenu ne vérifie pas les contraintes de ressources, la recherche est poursuivie jusqu'à ce qu'une autre feuille soit atteinte. Comme dans notre stratégie de recherche, une explication doit être calculée à chaque fois qu'un conflit apparaît, il est difficile de déterminer une explication lorsque l'ordonnancement ainsi obtenu ne vérifie pas les contraintes de ressources.

Il est donc préférable d'avoir une stratégie pour laquelle un ordonnancement réalisable (s'il existe) (*i.e.* toutes les disjonctions sont orientées et toutes les contraintes de ressources sont respectées) est obtenu lorsque la recherche atteint une feuille de l'arbre de recherche.

Nous avons donc décidé de vérifier les contraintes de ressources en chaque nœud de l'arbre de recherche, et d'adapter ce schéma de séparation à nos besoins. Pour cela, nous avons développé le schéma de séparation suivant [Elkhyari *et al.*, 2002b, Elkhyari *et al.*, 2002c] :

Chaque nœud de l'arbre de recherche est représenté par un triplet *(C, P, F)* où :

– C est l'ensemble des couples d'activités qui sont en conjonction (liés par des contraintes

de précédence). Il est initialisé par les couples d'activités qui sont reliées par les contraintes de précédence.

- P est l'ensemble des couples d'activités qui se chevauchent. Cet ensemble est initialisé par les couples d'activités en parallèle obtenus par propagation des contraintes temporelles.
- F est l'ensemble des couples d'activités n'appartenant ni à C, ni à P.

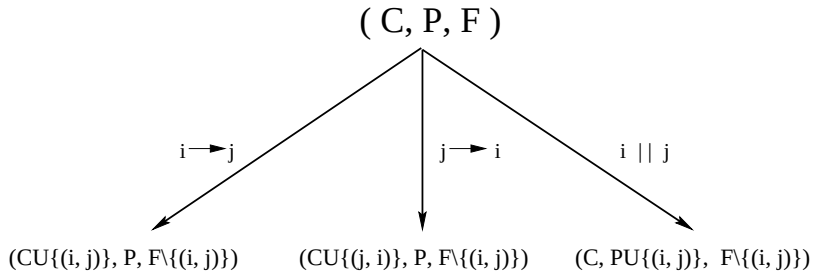


FIG. 7.1 – *Schéma de séparation.*

La stratégie consiste à sélectionner un couple d'activités (i, j) de F et à créer les trois fils f_1, f_2, f_3 suivants :

- $f_1 = (C \cup \{(i,j)\}, P, F \setminus \{(i,j)\})$, qui correspond à fixer la conjonction $i \rightarrow j$.
- $f_2 = (C \cup \{(j,i)\}, P, F \setminus \{(i,j)\})$, dans lequel nous fixons la conjonction $j \rightarrow i$.
- $f_3 = (C, P \cup \{(i,j)\}, F \setminus \{(i,j)\})$, dans lequel nous imposons la relation de parallélisme $i \parallel j$.

Dans le but d'intégrer cette stratégie dans notre système, nous avons proposé une nouvelle modélisation, basée sur la *notion de distance*, pour traduire les décisions prises en chaque nœud ainsi que leur négations en contraintes simples.

7.2 Notion de distance

Après avoir présenté les difficultés rencontrées pour la résolution des problèmes comportant les opérateurs $\vee$ et $\wedge$, nous présentons quelques approches de la littérature permettant de les résoudre. Nous présentons la façon dont les activités sont représentées dans notre système. En suite, nous définissons la notion de distance et présentons la modélisation des contraintes de décision et leur négations.

7.2.1 Préambule

Les contraintes de décision constituant notre schéma de séparation peuvent être considérées comme des contraintes temporelles. La plupart de ces contraintes font appel aux opérateurs $\vee$ et $\wedge$. Citons par exemple la contrainte disjonctive $i \leftrightarrow j$ qui se modélise par la

disjonction : $(t_j \geq t_i + p_i) \vee (t_i \geq t_j + p_j)$, ou des contraintes temporelles généralisées que nous verrons par la suite et qui s'expriment sous forme de disjonction de contraintes (*cf.* chapitre 11 page 152). Or ces opérateurs ($\vee$ et $\wedge$) ne sont pas faciles ni à mettre en œuvre, ni à manipuler dans un système de contraintes. En effet, pour les problèmes définis par une disjonction de contraintes, comme les problèmes d'ordonnancement d'ateliers (*Job-shop, Open-shop et Flow-shop*), *DeBacker* [1995] a montré que de tels problèmes introduisent une forte combinatoire.

7.2.2 Procédures de résolution

Une approche naïve, pour la résolution des problèmes définis par une disjonction des contraintes, consiste à considérer une disjonction comme un simple point de choix. L'utilisation d'une telle approche construit un nombre exponentiel de systèmes de contraintes à résoudre. En plus, nous sommes parfois amenés à résoudre tous ces systèmes de contraintes afin de déterminer une meilleure solution.

D'autres approches ont été proposées dans la littérature. Nous citons par exemple le mécanisme de *démon* de CHIP [Dincbas *et al.*, 1988], ou l'utilisation de l'opérateur de cardinalité [Van Hentenryck et Deville, 1991], où le point de choix est reporté jusqu'à ce qu'une décision puisse être prise. Une telle approche n'est intéressante que lorsque les autres contraintes du système permettent de prendre des décisions assez vite et elle est moins intéressante lorsque le nombre de contraintes disjonctives est grand. Ou encore, des approches qui considérent une disjonction comme un cas particulier de contrainte globale. Nous citons par exemple la contrainte cumulative de CHIP [Aggoun et Baldiceanu, 1993], d'autres exemples sont présentés dans [Boizumault *et al.*, 1995, Boizumault *et al.*, 1996]. Notons que l'utilisation de la contrainte cumulative reste limitée à un cadre d'*allocation de ressource*. Une approche plus intéressante consiste à introduire une contrainte ONE-OF [Jussien, 1997a] qui permet de traiter la disjonction exclusive de contraintes quelconques. Ces contraintes tentent chacune d'installer leur première contrainte. Lorsqu'un échec se produit, un point de retour est identifié et la contrainte ONE-OF est mise en cause et installe sa contrainte suivante. Signalons que l'utilisation des *explications* permet de ne pas défaire le travail indépendant entre les deux étapes. Cette approche a été utilisée pour la résolution des problèmes d'*Open-shop* [Guéret *et al.*, 1998].

Dans le chapitre 11, nous présentons une modélisation de quelques contraintes temporelles généralisées sans avoir recours à aucune approche pour la résolution des problèmes définis par une disjonction des contraintes.

7.2.3 Représentation des activités

La représentation que nous avons choisie est la plus simple et la plus utilisée pour la formulation des problèmes d'ordonnancement dans un système de résolution basé sur les contraintes. Notons qu'il existe d'autres formulations plus complexes comme celle proposée par [Smith et Cheng, 1993, Cheng et Smith, 1994].

Une variable t_i est associée à chaque activité i. Cette variable représente la date de démarrage de l'activité i. À chaque activité i est associée sa date de début au plus tôt r_i et sa date de début au plus tard f_i qui représentent respectivement la borne inférieure et supérieure de son domaine.

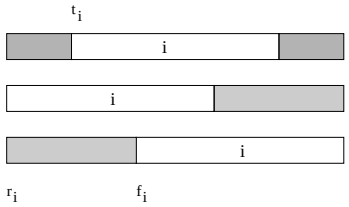


FIG. 7.2 – *Représentation d'une activité.*

Exemple 15 (Variables associées aux activités) :
Considérons un problème d'ordonnancement comportant quatre activités $\{1, 2, 3, 4\}$, et soit $UB = 7$ une borne supérieure de la date de fin de l'ordonnancement.
Au début tous les r_i sont nuls et les f_i sont égaux à $UB - p_i$. Le tableau 7.1 fourni les variables créées et leurs domaines.

Activités	p_i	**Variables**	**Domaines**
1	4	t_1	$[0, 3]$
2	2	t_2	$[0, 5]$
3	1	t_3	$[0, 6]$
4	3	t_4	$[0, 4]$

TAB. 7.1 – *Variables créées dans l'exemple 15*

7.2.4 Définition d'une distance

Brucker et al. [1998] ont introduit la notion de distance dans le but de détecter des conflits au cours d'une recherche arborescente. Nous en proposons ici une extension [Elkhyari *et al.*, 2002b, Elkhyari *et al.*, 2002c].

Définition 10 (Distance)
Soient i et j deux activités distinctes.
Nous définissons la distance d_{ij} entre i et j comme la quantité de temps s'écoulant entre la date de fin de i et la date de début de j.
Nous avons $d_{ij} = t_j - t_i - p_i$ (Fig. 7.3). On notera qu'il s'agit d'une notion algébrique.

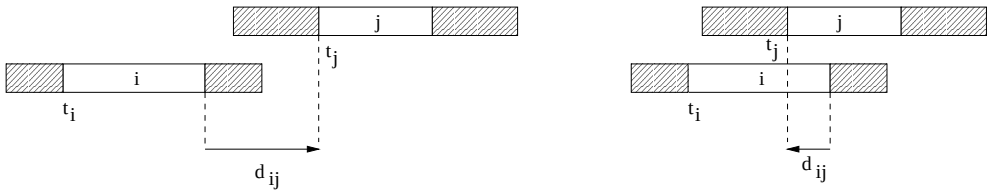


FIG. 7.3 – *Distance entre deux activités.*

De cette notion nous pouvons déduire les propriétés suivantes :

- la distance minimale entre i et j est $d_{ij}^{min} = r_j$ - f_i - p_i.
- la distance maximale entre i et j est $d_{ij}^{max} = f_j$ - r_i - p_i.
- $d_{ji} = -d_{ij} - p_i - p_j$.
- $d_{jk} = d_{ik} - d_{ij} - p_j$.

7.2.5 Distance et schéma de séparation

Nous pouvons déterminer les positions relatives entre deux activités i et j suivant les valeurs algébriques possibles de la distance d_{ij} (Fig. 7.4). Ainsi, nous avons :

- $d_{ij} \geq 0$ si et seulement si i précède j.
- $d_{ij} \leq -p_i - p_j$ si et seulement si i succède j (*i. e.* j précède i).
- $-p_i - p_j < d_{ij} < 0$ si et seulement si i et j sont partiellement en parallèle.

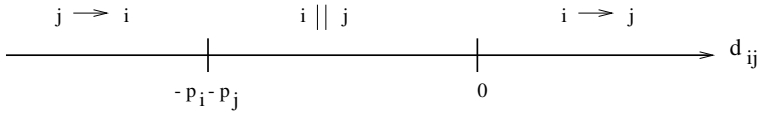


FIG. 7.4 – *Positions relatives de deux activités i et j en fonction de la distance d_{ij}.*

Nous remarquons que d_{ij} et d_{ji} ne peuvent pas prendre deux valeurs strictement positives en même temps. Ce qui entraîne qu'une condition nécessaire et suffisante pour que i et j soient partiellement en parallèle est que $d_{ij} \times d_{ji} > 0$.

Le tableau 7.2 fournit la modélisation pour chacune des positions relatives possibles sous forme d'une contrainte portant sur la distance.

Position	Contrainte
$i \rightarrow j$	$d_{ij} \geq 0$
$i \parallel j$	$d_{ij} \times d_{ji} > 0$
$i \leftrightarrow j$	$d_{ij} \times d_{ji} \leq 0$
$i \nrightarrow j$	$d_{ij} < 0$

TAB. 7.2 – *Modélisation des positions relatives*

Notre schéma de séparation est alors facilement adaptable à la stratégie de recherche. En effet, les trois décisions prises en chaque nœud de l'arbre de recherche sont modélisables en contraintes simples :

- le fils f_1 correspond à l'ajout de la contrainte $d_{ij} \geq 0$,
- le fils f_2 est équivalent à ajouter la contrainte $d_{ji} \geq 0$,

Décision	Contrainte	Négation de la décision	Négation de la contrainte
$i \rightarrow j$	$d_{ij} \geq 0$	$i \nrightarrow j$	$d_{ij} < 0$
$i \parallel j$	$d_{ij} \times d_{ji} > 0$	$i \leftrightarrow j$	$d_{ij} \times d_{ji} \leq 0$

TAB. 7.3 – *Traduction des décisions sous forme de contraintes*

– et le fils f_3 correspond à ajouter la contrainte $d_{ij} \times d_{ji} > 0$.

De plus, les négations de ces décisions sont facilement exprimables sous forme de contraintes simples (*cf.* tableau 7.3).

Notons que lorsque le fils d'un nœud *(C, P, F)* est créé, la contrainte correspondante est ajoutée au système. Par propagation de cette contrainte, de nouvelles positions relatives entre des activités peuvent être déduites. Des couples d'activités de l'ensemble F peuvent donc être transférés vers les ensembles C ou P. De même, lorsqu'une contrainte est retirée du système et après propagation, des couples d'activités peuvent être transférés des ensembles C et P vers l'ensemble F.

7.2.6 Représentation des distances

Dans notre système, les distances sont représentées par des variables. À chaque distance d_{ij} est associée son activité de début i, son activité de fin j, et son domaine $D_{ij} = [d_{ij}^{min}, d_{ij}^{max}]$ où $d_{ij}^{min} = r_j - f_i - p_i$ et $d_{ij}^{max} = f_j - r_i - p_i$.

Lorsqu'une distance est créée, la contrainte $C_{ij} : d_{ij} = t_j - t_i - p_i$, appelée *contrainte de distance*, est ajoutée au système. Elle permet de maintenir la correspondance entre activités et distances. Les variables d_{ij} ne sont créées qu'en cas de besoin.

Exemple 16 (Variables associées aux distances) :
Reprenons l'exemple 15, et supposons que les décisions suivantes ont été prises au cours de la recherche : $1 \rightarrow 2$, $1 \rightarrow 3$, $2 \parallel 4$, et $3 \leftrightarrow 4$. Nous créons les distances d_{12}, d_{13}, d_{24}, d_{42}, d_{34}, et d_{43}. À chacune de ces distances d_{ij} est associé son domaine $D_{ij} = [-UB, UB - p_i - p_j]$. Le tableau 7.4, produit les distances introduites, leur domaines, les contraintes de distance associées et les contraintes de décision.

Distances	Domaines	Contraintes de distance	Contraintes de décision	Modélisation
d_{12}	[-7, 1]	$C_{12} : d_{12} = t_2 - t_1 - 4$	$1 \rightarrow 2$	$d_{12} \geq 0$
d_{13}	[-7, 2]	$C_{13} : d_{13} = t_3 - t_1 - 4$	$1 \rightarrow 3$	$d_{13} \geq 0$
d_{24}	[-7, 2]	$C_{24} : d_{24} = t_4 - t_2 - 2$		
d_{42}	[-7, 2]	$C_{42} : d_{42} = t_2 - t_4 - 3$	$2 \parallel 4$	$d_{24} \times d_{42} > 0$
d_{34}	[-7, 3]	$C_{34} : d_{34} = t_4 - t_3 - 1$		
d_{43}	[-7, 3]	$C_{43} : d_{43} = t_3 - t_4 - 3$	$3 \leftrightarrow 4$	$d_{34} \times d_{43} \leq 0$

TAB. 7.4 – *Distances et contraintes créées dans l'exemple 16*

7.3 Critère de sélection

Nous avons testé plusieurs critères pour sélectionner le couple d'activités *(i, j)* de F en chaque nœud de l'arbre de recherche :

- le couple d'activités pour lequel la taille du domaine de la variable d_{ij} associée est minimale;
- le couple d'activités dont le rapport entre la taille du domaine de d_{ij} et le nombre de contraintes utilisant cette variable est minimal;
- le couple d'activités dont la somme des durées opératoires $p_i + p_j$ est maximale;
- et le couple d'activités dont la somme des énergies $w_{ik} + w_{jk}$ sur une des ressources k de R est maximale.

Nous avons retenu le dernier critère car il donne de meilleurs résultats que les autres (sans doute parce que ce critère permet de détecter des conflits de ressources bien avant les autres).

Notons que le nombre de branches à explorer en un nœud *(C, P, F)* peut être réduit. En effet, lorsqu'un couple *(i, j)* de F est sélectionné, nous pouvons connaître les différentes positions relatives possibles entre ses deux activités :

- si le domaine de la variable d_{ij} est $[u, v]$ où $-p_i - p_j < u < 0 < v$, alors soit i précède j, soit i et j sont partiellement en parallèle. Donc ce nœud n'aura pas de fils de type f_2.
- si le domaine de la variable d_{ij} est $[u, v]$ où $u < -p_i - p_j < v < 0$, alors soit j précède i, soit i et j sont partiellement en parallèle. Ce nœud n'aura donc pas de fils de type f_1.

Dans notre système, nous avons utilisé la règle de déduction qui résulte de la capacité de la ressource. Son principe est le suivant : si pour une ressource k, la somme des besoins en ressource $a_{ik} + a_{jk}$ de deux activités i et j dépasse la capacité de la ressource R_k alors nous sommes sûrs que ces deux activités ne peuvent pas être éxecutées en même temps. Ce qui correspond à l'ajout de la contrainte disjonctive $d_{ij} \times d_{ji} \leq 0$. Cette règle permet aussi de réduire le nombre de branches à explorer au cours de la recherche.

En ce qui concerne le critère de choix des fils, nous suivons la procédure suivante : lorsque nous sélectionnons un couple *(i, j)* de F, nous déterminons la liste des fils possibles de ce nœud ordonnée suivant l'ordre f_1, f_2 puis f_3. L'exploration des branches suit donc l'ordre de cette liste.

7.4 Borne supérieure

Dans le chapitre 2, nous avons présenté les principales approches pour le calcul des bornes inférieures et supérieures pour le RCPSP. Pour la borne inférieure, les approches les plus performantes sont difficles à mettre en œuvre et très coûteuses en temps de calcul. Par contre, pour la borne supérieure, les approches les plus utilisées sont celles qui sont basées sur des règles de priorités. Ces approches sont faciles à mettre en œuvre, ne nécessitent pas beaucoup de temps de calcul, et sont plus ou moins performantes.

Dans notre système, nous avons utilisé une heuristique simple basée sur les règles de priorité (*cf.* 2.3.1 page 10). Elle utilise le schéma de génération d'ordonnancements en série

Algorithme 5: L'algorithme de la résolution optimale utilisé dans notre système.

Données : un problème et sa borne supérieure ub.
Résultat : une solution optimale du problème.

```
Minimise_Cmax(pb : problème, ub : entier)
début
    dernière_solution ← échec
    arrêt ← faux
    répéter
        pb ← pb ∪ (Cmax < ub)
1       solution ← répare_solution(pb)
        si solution = échec alors
            arrêt ← vrai
        sinon
            ub ← valeur du Cmax de la solution
            dernière_solution ← solution
    jusqu'à arrêt = vrai
    retourner dernière_solution
fin
```

et les trois règles de priorité présentées dans le tableau 7.5. Rappelons que l'ensemble E_g, déterminé à chaque itération de l'algorithme, représente l'ensemble des activités dont tous les prédécesseurs ont été ordonnancés. Et S_i est l'ensemble des successeurs de l'activité i.

	Règle	Activité sélectionnée
GTP	greatest processing time	$\max_{i \in E_g}\{p_i\}$
MIS	most immediate successors	$\max_{i \in E_g}\{\|S_i\|\}$
GE	greatest energy	$\max_{k \in R}\{\max_{i \in E_g}\{w_{ik}\}\}$

TAB. 7.5 – *Règles de priorité*

Nous avons implémenté ces trois règles. La borne supérieure choisie est la meilleure solution obtenue par ces trois heuristiques.

7.5 Recherche d'une solution optimale

Dans le chapitre 4 (*cf.* section 4.3.2 page 41), nous avons présenté deux algorithmes permettant de résoudre les problèmes d'optimisation sous contraintes. Nous en avons retenu le premier algorithme car il n'utilise que la borne supérieure. L'algorithme utilisé dans notre système est une version amélioré de celui ci. Au lieu de recommencer la recherche depuis le début, l'algorithme se contente de réparer l'ancienne solution.

Ainsi, si nous considérons un problème d'ordonnancement dont la fonction objectif est de minimiser la date de fin du projet (C_{max}), la procédure permettant de le résoudre optimalement est décrite dans l'algorithme 5. La fonction *répare_solution*(*pb*) cherche à trouver une solution du nouveau problème en réparant la solution obtenue à l'étape précédente.

7.6 Conclusion

Dans ce chapitre, nous avons présenté quelques outils pour la conception de notre système. Ainsi, nous avons développé un schéma de séparation, donné une modélisation des contraintes de décision en contraintes simples, et présenté un algortithme pour la recherche d'une solution optimale.

Il nous reste donc à integrer les contraintes temporelles (contraintes de précédence, contraintes de parallélisme ou de chevauchement, et contraintes disjonctives) dans notre système (*cf.* chapitre 8), à modéliser les contraintes de ressources en contraintes globales et à les intégrer dans notre système (*cf.* chapitre 9).

Chapitre 8

Contraintes temporelles

Sommaire

Introduction

Nous nous intéressons essentiellement dans ce chapitre aux contraintes temporelles simples telles que rencontrées dans le RCPSP classique, ou utilisées dans la méthode de résolution que nous allons utiliser. Nous traiterons ultérieurement les extensions de ces contraintes (*cf.* chapitre 11). Nous présentons aussi dans ce chapitre, le mécanisme de propagation de ces contraintes, et nous montrons comment produire des explications lors de leur propagation. Nous supposons ici que les durées opératoires des activités sont constantes.

8.1 Rappel et exemple illustratif

Nous avons présenté, dans le chapitre 7, une modélisation basée sur la notion de distance permettant d'exprimer des contraintes temporelles (précédence, chevauchement ou parallélisme, disjonction, et la négation de la précédence) en contraintes simples. Le tableau 8.1 rappele la modélisation de ces contraintes.

Contrainte	Modélisation
$i \rightarrow j$	$d_{ij} \geq 0$
$i \parallel j$	$d_{ij} \times d_{ji} > 0$
$i \leftrightarrow j$	$d_{ij} \times d_{ji} \leq 0$
$i \nrightarrow j$	$d_{ij} < 0$

TAB. 8.1 – *Modélisation des containtes temporelles*

Nous avons aussi présenté, dans le même chapitre, la représentation des activités et des distances dans notre système. Une activité i est représentée par une variable t_i associée à la date de début de cette activité. Et une distance entre deux activités i et j est représentée par une variable d_{ij}. Notons que lorsqu'une distance est créée, la *contrainte de distance* C_{ij} : $d_{ij} = t_j - t_i - p_i$ est automatiquement ajoutée au système, et que les variables d_{ij} ne sont créées qu'en cas de besoin (*cf.* exemple 17).

Exemple 17 (Exemple illustratif) :
Considérons un problème d'ordonnancement comportant quatre activités de durées opératoires respectivement $p_1 = 4$, $p_2 = 2$, $p_3 = 1$, et $p_4 = 3$. Soit $UB = 7$ une borne supérieure de la date de fin de l'ordonnancement. Supposons que les quatre activités sont liées par les contraintes temporelles : $1 \rightarrow 2$, $1 \rightarrow 3$, $2 \parallel 4$, et $3 \leftrightarrow 4$.
Ce problème peut être modélisé comme un CSP :
Les variables : t_1, t_2, t_3, t_4, d_{12}, d_{13}, d_{24}, d_{42}, d_{34}, d_{43}.
Les domaines : $D_1 = [0,3]$, $D_2 = [0,5]$, $D_3 = [0,6]$, $D_4 = [0,4]$, $D_{12} = [-7,1]$, $D_{13} = [-7,2]$, $D_{24} = [-7,2]$, $D_{42} = [-7,2]$, $D_{34} = [-7,3]$, $D_{43} = [-7,3]$.
Les contraintes : $d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, $C_{12} : d_{12} = t_2 - t_1 - 4$, $C_{13} : d_{13} = t_3 - t_1 - 4$, $C_{24} : d_{24} = t_4 - t_2 - 2$, $C_{42} : d_{42} = t_2 - t_4 - 3$, $C_{34} : d_{34} = t_4 - t_3 - 1$, $C_{43} : d_{43} = t_3 - t_4 - 3$.

8.2 Propagation des contraintes temporelles

Comme nous l'avons vu au chapitre 5, la bibliothèque PaLM qui constitue l'environnement de programmation de notre système, est basée sur un modèle à événements. Au cours de la propagation, chaque fois que le domaine d'une variable est réduit, les contraintes utilisant cette variable sont réveillées et peuvent générer d'autres nouveaux événements.

Dans notre système, la réduction des domaines se fait par modification des bornes des domaines. Le filtrage se fait donc par des appels aux deux méthodes `awakeOnInf` et `awakeOnSup`.

Dans cette section, nous nous intéressons uniquement au mécanisme de propagation des contraintes temporelles. Le calcul des explications associées à ces contraintes sera présenté dans la section 8.3.

8.2.1 Contrainte de précédence

La contrainte de précédence entre deux activités i et j est représentée par la contrainte unaire $d_{ij} \geq 0$. Cette contrainte est une des contraintes de base de la bibliothèque CHOCO. Le domaine D_{ij} de la variable d_{ij} est remplacé par $D_{ij} = [\max\{d_{ij}^{min}, 0\}, d_{ij}^{max}]$.

Exemple 18 (Contraintes de précédence) :
Si nous reprenons l'exemple 17, la propagation de la contrainte $d_{12} \geq 0$ transforme le domaine $D_{12} = [-7, 1]$ en $D_{12} = [0, 1]$.
De même, à cause de la contrainte $d_{13} \geq 0$, le domaine $D_{13} = [-7, 2]$ de d_{13} devient $D_{13} = [0, 2]$.

8.2.2 Contrainte de distance

Les contraintes de distance sont des contraintes linéaires. Le mécanisme de propagation des contraintes linéaires existe déjà dans la bibliothèque CHOCO. Nous nous contentons de donner quelques exemples.

Exemple 19 (Propagation des contraintes de distance) :
Reprenons l'exemple 18.
Du fait que le domaine de d_{12} devient $[0, 1]$, la propagation de la contrainte $C_{12} : d_{12} = t_2 - t_1 - 4$ ajuste les domaines des variables t_1 et t_2 respectivement à [0, 1] et [4, 5].
De même, comme le domaine de d_{13} devient $[0, 2]$, la propagation de la contrainte $C_{13} : d_{13} = t_3 - t_1 - 4$ ajuste le domaine de la variable t_3 à [4, 6]. Ceci entraîne, à cause de la contrainte $C_{34} : d_{34} = t_4 - t_3 - 1$, l'ajustement du domaine de la variable d_{34} à $D_{34} = [-7, -1]$.

8.2.3 Contrainte exprimant la disjonction

Comme nous l'avons vu précédement, deux activités i et j sont en disjonction si et seulement si leurs distances d_{ij} et d_{ji} vérifient la contrainte $d_{ij} \times d_{ji} \leq 0$. Cette contrainte est introduite comme une contrainte binaire utilisant les deux variables d_{ij} et d_{ji}. Son mécanisme de propagation est comme suit :

– si la borne inférieure de la variable d_{ij} (*resp.* d_{ji}) devient strictement positive alors la

Algorithme 6: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $d_{ij} \times d_{ji} \leq 0$ et d représente une de ses variables

awakeOnInf(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{min} > 0$ **alors**
1     $updateSup(d_{ji}, 0)$
  si $d = d_{ji}$ **alors**
    si $d_{ji}^{min} > 0$ **alors**
      $updateSup(d_{ij}, 0)$
fin

Algorithme 7: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $d_{ij} \times d_{ji} \leq 0$ et d représente une de ses variables

awakeOnSup(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{max} < 0$ **alors**
1     $updateInf(d_{ji}, 0)$
  si $d = d_{ji}$ **alors**
    si $d_{ji}^{max} < 0$ **alors**
      $updateInf(d_{ij}, 0)$
fin

borne supérieure de la variable d_{ji} (*resp.* d_{ij}) est ajustée à la valeur 0 (Algorithme 6).

– si la borne supérieure de la variable d_{ij} (*resp.* d_{ji}) devient strictement négative alors la borne inférieure de la variable d_{ji} (*resp.* d_{ij}) est ajustée à la valeur 0 (Algorithme 7).

Notons que les méthodes `updateSup` (ligne 1 dans l'algorithme 6) et `updateInf` (ligne 1 dans l'algorithme 7) ne modifient respectivement la borne supérieure et inférieure de la variable d_{ji} que si cette borne a effectivement changée.

Exemple 20 (Contrainte de disjonction) :
Reprenons l'exemple 19.
Comme le domaine de la variable d_{34} devient $[-7, -1]$, la propagation de la contrainte $d_{34} \times d_{43} \leq 0$ ajuste le domaine de la variable d_{43} à $[0, 3]$. Ceci entraîne par propagation de la contrainte $C_{43} : d_{43} = t_3 - t_4 - 3$ l'ajustement du domaine D_4 de la variable t_4 à $[0, 3]$.
Du fait que le domaine de t_4 devient $[0, 3]$, la propagation de la contrainte $C_{24} : d_{24} = t_4 - t_2 - 2$ ajuste le domaine de la variable d_{24} à $[-7, -3]$.

Algorithme 8: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $d_{ij} \times d_{ji} > 0$ et d représente une de ses variables

awakeOnInf(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{min} > 0$ **alors**
      $updateInf(d_{ji}, 1)$
  si $d = d_{ji}$ **alors**
    si $d_{ji}^{min} > 0$ **alors**
      $updateInf(d_{ij}, 1)$
fin

Algorithme 9: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $d_{ij} \times d_{ji} > 0$ et d représente une de ses variables

awakeOnSup(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{max} < 0$ **alors**
      $updateSup(d_{ji}, -1)$
  si $d = d_{ji}$ **alors**
    si $d_{jj}^{max} < 0$ **alors**
      $updateSup(d_{ij}, -1)$
fin

8.2.4 Contrainte exprimant le parallélisme

Deux activités i et j sont partiellement en parallèle si et seulement si leurs distances d_{ij} et d_{ji} vérifient la contrainte $d_{ij} \times d_{ji} > 0$. Cette contrainte est déclarée comme une contrainte binaire sur les deux variables d_{ij} et d_{ji}. Le principe de propagation de cette contrainte est comme suit :

- si la borne inférieure de la variable d_{ij} (*resp.* d_{ji}) devient strictement positive alors la borne inférieure de la variable d_{ji} (*resp.* d_{ij}) est ajustée à la valeur 1 (Algorithme 8).

- si la borne supérieure de la variable d_{ij} (*resp.* d_{ji}) devient strictement négative alors la borne supérieure de la variable d_{ji} (*resp.* d_{ij}) est ajustée à la valeur -1 (Algorithme 9).

Exemple 21 (Contrainte de parallélisme) :
Reprenons l'exemple 20. Du fait que le domaine de la variable d_{24} devient [-7, -3], la propagation de la contrainte $d_{24} \times d_{42} > 0$ ajuste le domaine D_{42} de la variable d_{42} à $[-7, -1]$.
Le tableau 8.2, nous donne les domaines des variables obtenus lorsque la propagation s'achève.
Dans la figure 8.1, nous représentons les domaines des activités avant la propagation (a) et ceux obtenus lorsque la propagation s'achève (b).

Variables	Domaines	Nouveaux domaines
t_1	[0, 3]	[0, 1]
t_2	[0, 5]	[4, 5]
t_3	[0, 6]	[5, 6]
t_4	[0, 4]	[2, 3]
d_{12}	[-7, 1]	[0, 1]
d_{13}	[-7, 2]	[0, 2]
d_{24}	[-7, 2]	[-4, -3]
d_{42}	[-7, 2]	[-2, -1]
d_{34}	[-7, 3]	[-5, -4]
d_{43}	[-7, 3]	[0, 1]

TAB. 8.2 – *Propagation des contraintes dans l'exemple 16*

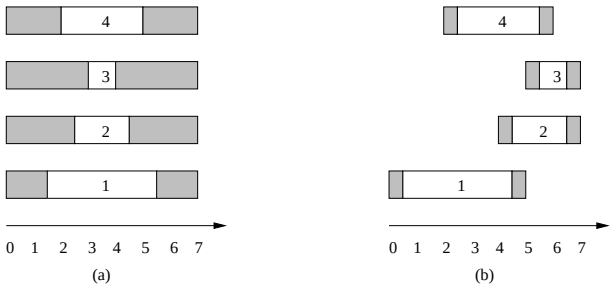


FIG. 8.1 – *Représentation graphique de l'exemple 21*

8.3 Calcul des explications

Comme nous nous plaçons dans un cadre dynamique, il est nécessaire d'introduire les explications dans le traitement des contraintes. L'introduction des explications dans les contraintes temporelles est facile, il suffit de modifier les méthodes `awakeOnInf` et `awakeOnSup` en ajoutant de nouvelles informations.

Le système PALM nous fournit des outils qui permettent d'enregistrer les explications pour chaque événement et de les récupérer. Nous rappelons que, dans PALM, la méthode `becauseOf` permet de construire les explications. Elle permet d'ajouter une explication à une

Algorithme 10: La borne inférieure de l'une des variables d_{ij} ou d_{ji} a augmenté.

C est une contrainte de type $d_{ij} \times d_{ji} \leq 0$ et d représente une de ses variables

awakeOnInf(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{min} > 0$ **alors**
1     $updateSup(d_{ji},\ 0,\ becauseOf(theConstraint(C),\ theInf(d_{ij})))$
  si $d = d_{ji}$ **alors**
    si $d_{ji}^{min} > 0$ **alors**
      $updateSup(d_{ij},\ 0,\ becauseOf(theConstraint(C),\ theInf(d_{ji})))$
fin

liste d'explications associées à l'information relatives à l'evénement comme `theSup(x)`, qui permet d'expliquer la borne supérieure de la variable x, ou `theInf(x)`, qui permet d'expliquer la borne inférieure de x (*cf.* section 5.5 page 56).

Nous signalons que dans le système PaLM, la gestion des explications pour les contraintes de précédence (contraintes unaires) et les contraintes de distance (contraintes linéaires) est déjà traitée.

8.3.1 Contrainte exprimant la disjonction

Pour cette contrainte, les informations à ajouter sont simples car toutes ces informations sont disponibles dans les codes `awakeOnInf` et `awakeOnSup`.

Pour la méthode `awakeOnInf`, la modification de la borne supérieure de la variable d_{ji} est due :

- à la contrainte elle-même, elle doit être ajoutée à l'explication calculée. Pour cela, la méthode `theConstraint(C)` est utilisée (ligne 1 dans l'algorithme 10).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode `theInf(`d_{ij}`)` (ligne 1 dans l'algorithme 10).

Pour la méthode `awakeOnSup`, la modification de la borne inférieure de la variable d_{ji} est due :

- à la contrainte elle-même, cette contrainte doit être ajoutée à l'explication calculée. Le méthode `Constraint(C)` est alors appliquée (ligne 1 dans l'algorithme 11).
- aux contraintes responsables de la modification de la borne supérieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode `theSup(`d_{ij}`)` (ligne 1 dans l'algorithme 11).

Notons que, dans la méthode `updateInf` (*resp.* `updateSup`), la méthode `becauseOf` ajoute l'explication `theConstraint(C)` aux explications associées à `theInf(`d_{ij}`)` (*resp.* `theSup(`d_{ij}`)`).

Algorithme 11: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $d_{ij} \times d_{ji} \leq 0$ *et d représente une de ses variables*

awakeOnSup(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{max} < 0$ **alors**
1     $updateInf(d_{ji},\ 0,\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
  si $d = d_{ji}$ **alors**
    si $d_{ji}^{max} < 0$ **alors**
      $updateInf(d_{ij},\ 0,\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
fin

Algorithme 12: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $d_{ij} \times d_{ji} > 0$ *et d représente une de ses variables*

awakeOnInf(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{min} > 0$ **alors**
1     $updateInf(d_{ji},\ 1,\ becauseOf(theConstraint(C),\ theInf(d_{ij})))$
  si $d = d_{ji}$ **alors**
    si $d_{ji}^{min} > 0$ **alors**
      $updateInf(d_{ji},\ 1,\ becauseOf(theConstraint(C),\ theInf(d_{ji})))$
fin

8.3.2 Contrainte exprimant le parallélisme

L'introduction des explications dans cette contrainte est simple, les informations dont nous avons besoin sont disponibles dans les méthodes `awakeOnInf` et `awakeOnSup`.

Pour la méthode `awakeOnInf`, la modification de la borne inférieure de la variable d_{ji} est due :

- à la contrainte elle-même, cette contrainte doit être ajoutée à l'explication calculée. Le méthode `Constraint(C)` est alors appliquée (ligne 1 dans l'algorithme 12).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode `theInf(`d_{ij}`)` (ligne 1 dans l'algorithme 12).

Pour la méthode `awakeOnSup`, la modification de la borne supérieure de la variable d_{ji} est due :

- à la contrainte elle-même, cette contrainte doitêtre ajoutée à l'explication calculée. Le méthode `Constraint(C)` est alors appliquée (ligne 1 dans l'algorithme 13).

Algorithme 13: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $d_{ij} \times d_{ji} > 0$ et d représente une de ses variables

awakeOnSup(C:**contrainte**, d:**distance**)
début
  si $d = d_{ij}$ **alors**
    si $d_{ij}^{max} < 0$ **alors**
1     $updateSup(d_{ji},\ -1,\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
  si $d = d_{ji}$ **alors**
    si $d_{jj}^{max} < 0$ **alors**
      $updateSup(d_{ji},\ -1,\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
fin

– aux contraintes responsables de la modification de la borne supérieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode `theSup`(d_{ij}) (ligne 1 dans l'algorithme 13).

8.3.3 Gestion des problèmes dynamiques

Comme nous le montre l'exemple 22, le calcul des explications pour les contraintes temporelles est devenu une tâche facile. Et donc, nous pouvons facilement ajouter et retirer incrémentalement des contraintes temporelles.

Exemple 22 (Calcul des explications) :
Dans l'exemple 19, à cause des deux contraintes: $d_{12} \leq 0$ et C_{12} : $d_{12} = t_2 - t_1 - 4$, les domaines des variables t_1 et t_2 sont ajustées respectivement à $[0, 1]$ et $[4, 5]$. L'explication de la modification de la borne supérieure du domaine de la variable t_1 sera donc `theSup`(t_1) $= \{d_{12} \leq 0, C_{12} : d_{12} = t_2 - t_1 - 4\}$. De même, l'explication de la modification de la borne inférieure du domaine de la variable t_2 sera `theInf`(t_2) $= \{d_{12} \leq 0, C_{12} : d_{12} = t_2 - t_1 - 4\}$.
Le tableau 8.3 reproduit les explications des modifications des bornes inférieures et supérieures des domaines des variables de l'exemple 16. Supposons que nous désirons retirer la contrainte $2 \parallel 4$: nous devons alors la retirer du réseau de contraintes. Puis, défaire ses effets passés en considérant toutes les explications contenant cette contrainte, ici se sont: `theInf`(t_3), `theInf`(d_{24}), `theDom`(d_{42}), `theInf`(d_{34}), et `theSup`(d_{43}). Les domaines de variables t_3, d_{24}, d_{42}, d_{34}, et d_{43} vont être alors rétablis. Et en fin, quelques valeurs rétablies vont être retirées en propageant de nouveau le problème.

Variable	Domaine	Explication des modifications borne inférieure (theInf)	Explication des modifications borne supérieure (theSup)
t_1	[0, 1]		$d_{12} \geq 0$, C_{12}
t_2	[4, 5]	$d_{12} \geq 0$, C_{12}	
t_3	[5, 6]	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{42}, C_{34}, C_{43}	
t_4	[2, 3]	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}	
d_{12}	[0, 1]	$d_{12} \geq 0$	
d_{13}	[1, 2]	$d_{13} \geq 0$	
d_{24}	[-4, -3]	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{34}, C_{43}	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{34}, C_{43}
d_{42}	[-2, -1]	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{42}, C_{34}, C_{43}	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{34}, C_{43}
d_{34}	[-5, -4]	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{42}, C_{34}, C_{43}	$d_{13} \geq 0$, $d_{34} \times d_{43} \leq 0$, C_{13}, C_{34}
d_{43}	[0, 1]	$d_{13} \geq 0$, $d_{34} \times d_{43} \leq 0$, C_{13}, C_{34}	$d_{12} \geq 0$, $d_{13} \geq 0$, $d_{24} \times d_{42} > 0$, $d_{34} \times d_{43} \leq 0$, C_{12}, C_{13}, C_{24}, C_{42}, C_{34}, C_{43}

TAB. 8.3 – *Explication des modifications des bornes inférieures et supérieures des domaines des variables (exemple 16).*

8.4 Conclusion

Dans ce chapitre, nous nous sommes intéressé aux contraintes temporelles simples tellesque : les contraintes de précédence, de chevauchement et de disjonction. Nous avons présenté le mécanisme de propagation et le calcul des explication des contraintes exprimant le chevauchement et la disjonction.

Dans notre système, ces contraintes sont utilisées pour exprimer :

- les contraintes temporelles du problème initial : pour le RCPSP, il s'agit des contraintes de précédence,
- les contraintes de décision prises au cours de la recherche : il s'agit des décisions définissant notre schéma de séparation ainsi que leur négations,
- et certains aléas pouvant intérvenir au cours ou après la résolution du problème.

Chapitre 9

Contraintes de ressources

Sommaire

Introduction

Nous avons vu dans le chapitre précédent que le mécanisme de propagation des contraintes temporelles et le calcul des explications associées se fait d'une manière assez simple.

Dans ce chapitre, nous allons traiter les contraintes de ressources. Nous rappelons que dans un RCPSP chaque activité i utilise une certaine quantité a_{ik} de ressource k tout au long de son exécution. Une contrainte de ressource spécifie que pour chaque intervalle de temps $[t-1, t)$, la capacité de la ressource doit être supérieure ou égale à la somme des besoins des activités en cours d'exécution. La contrainte de ressource peut être alors modélisée comme suit :

$$\forall k \in R,\ \forall t,\ \sum_{r_i \leq t < d_i} a_{ik} \leq R_k$$

Cette contrainte est difficile à traiter car elle nécessite la connaissance des dates de début de chacune des activités, or ceci est le but final du problème. Différents algorithmes permettant la propagation des contraintes de ressources ont été introduits. Nous en distinguons deux familles :

- celles qui utilisent uniquement les dates de début des activités : c'est le cas des *ensembles interdits* (*Schäffter* [1997]), des *parties obligatoires* (*Klein et Scholl* [1999]) ou de l'*histogramme* (*Caseau et Laburthe* [1994]),
- et celles qui utilisent la notion d'énergie : comme le *raisonnement énergétique* (*Erschler et lopez* [1990]), ou les *intervalles de tâches* (*Caseau et Laburthe* [1994]).

Pour les contraintes de ressources, nous avons choisi de traduire quelques règles de déductions en contraintes globales, (le calcul des ensembles interdits est coûteux et les explications associées sont très difficiles à produire). Nous avons utilisé les techniques suivantes : le principe des *parties obligatoires*, de l'*histogramme*, et les règles déduites de la notion d'*intervalle de tâches*.

L'utilisation des *parties obligatoires* a deux avantages : premièrement, leur mise en œuvre est très simple, et deuxièmement, la complexité est faible. Leur inconvénient est qu'elles ne sont efficaces que lorsque les fenêtres temporelles des activités deviennent petites.

Quant aux *intervalles de tâches*, ils sont efficaces et de complexité polynomiale. Mais leur mise en œuvre est assez difficile.

Nous allons donc présenter deux contraintes globales permettant la maintenance des contraintes de ressources [Elkhyari *et al.*, 2002a] :

- une contrainte basée sur les deux principes : *parties obligatoires* et *histogramme*,
- et une autre basée sur les *intervalles de tâches*.

Nous proposerons quelques améliorations des règles de déductions. Nous verrons en détail le mécanisme de propagation de ces contraintes et comment calculer les explications associées.

9.1 Représentation d'une ressource

Nous associons à chaque ressource sa capacité et l'ensemble des activités utilisant cette ressource.

Dans la suite, nous noterons $\prec t+1 \succ$ la période de temps définie par les deux instants t et $(t+1)$.

9.2 Contrainte basée sur les parties obligatoires

Dans cette section, nous présentons la contrainte globale combinant la notion de partie obligatoire [Klein et Scholl, 1999], et les techniques utilisant l'histogramme présenté dans [Caseau et Laburthe, 1994].

9.2.1 Parties obligatoires : rappel

Nous rappelons que l'idée de la partie obligatoire (*cf.* section 2.2.2 page 9) est d'associer à chaque activité i l'intervalle durant lequel une partie de cette activité est obligatoirement exécutée. La *partie obligatoire* $PO(i)$ sera donc définie par :

$$PO(i) = \begin{cases} [f_i, r_i + p_i] & si\ f_i < r_i + p_i, \\ \emptyset & sinon. \end{cases}$$

Exemple 23 (Parties obligatoires) :
Considérons un problème d'ordonnancement comportant trois activités $\{1, 2, 3\}$ et une seule ressource. Nous associons à chaque activité i, sa date de début au plus tôt r_i, sa date de début au plus tard f_i, et sa quantité de ressource a_{i1}. Le tableau 9.1 fournit les données du problème.
La figure 9.1 représente les parties obligatoires $PO(i)$ associées à chacune des activités du problème.

i	p_i	r_i	f_i	a_{i1}
1	4	2	4	3
2	4	1	2	2
3	5	2	4	4

TAB. 9.1 – *Données de l'exemple 23*

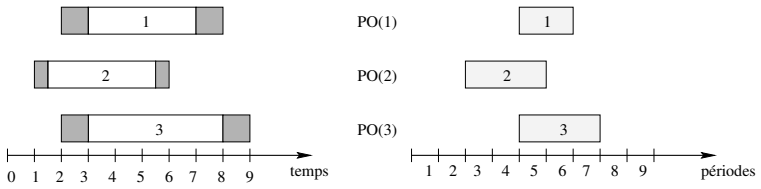


FIG. 9.1 – *Parties obligatoires.*

9.2.2 Modélisation

Lorsque la partie obligatoire $PO(i)$ associée à une activité i est non vide, nous réservons la quantité de ressource a_{ik} durant les périodes de temps $\prec f_i + 1 \succ, \cdots, \prec r_i + p_i \succ$ pour chaque ressource k utilisée par cette activité. Pour cela, nous associons à chaque ressource k une contrainte $\mathcal{C}_{po}(k)$ définie par les deux histogrammes suivants :

– un *histogramme des niveaux* représentant le niveau de ressource réservé pour chaque

période de temps $\prec t \succ$. Cet histogramme est représenté par une liste $\mathcal{H}n_k$, où $\mathcal{H}n_k[t]$ est égale à la quantité de ressource réservée sur la ressource k à cette période de temps $\prec t \succ$.

- un *histogramme des activités* qui stocke, pour chaque période de temps $\prec t \succ$, l'ensemble des activités pour lesquelles une quantité de ressource a été réservée. Cet histogramme est représenté par une liste $\mathcal{H}a_k$, où $\mathcal{H}a_k[t]$ est l'ensemble des activités pour lesquelles une quantité de ressource k a été réservée à la période de temps $\prec t \succ$.

Exemple 24 (Histogrammes des niveaux et histogrammes des activités) :
Reprenons l'exemple précédent et considérons la période de temps $\prec 5 \succ$. Nous constatons que les trois activités doivent être réalisées obligatoirement dans cette période de temps. Nous réservons la quantité de ressource $a_{11} + a_{21} + a_{31} = 9$ dans l'histogramme des niveaux ($\mathcal{H}n_1[5] = 9$), et nous conservons dans l'hisogramme des activités les trois activités 1, 2, et 3 ($\mathcal{H}a_1[5] = \{1,2,3\}$).
Pour l'exemple précédent, l'histogramme des niveaux sera représenté par la liste $\mathcal{H}n_1 = (0,0,2,2,9,7,4,0,0)$ et l'histogramme des activités par la liste $\mathcal{H}a_1 = (\{\},\{\},\{2\},\{2\},\{1,2,3\},\{1,3\},\{3\},\{\},\{\})$.

9.2.3 Maintien des structures de données

Une contrainte *parties obligatoires* $\mathcal{C}_{po}(k)$, sur une ressource k, est définie comme une contrainte globale. Les variables associées à cette contrainte sont toutes les variables t_i associées aux activités i qui utilisent cette ressource.

Comme nous nous situons dans un cadre de propagation basé sur la notion d'événements, nous avons besoin d'un système permettant de gérer la propagation des événements. Les événements que nous pouvons rencontrer lors de la propagation sont l'augmentation ou la diminution de la borne inférieure ou de la borne supérieure de la variable associée à une des activités.

À chaque fois qu'un de ces événements se produit, les histogrammes des niveaux et les histogrammes des activités sont mis à jour. Nous présentons dans ce qui suit, le principe de cette mise à jour.

9.2.3.1 La borne inférieure de la variable associée à une activité augmente

Si la date au plus tôt d'une activité i augmente de r_i à r_i', trois cas sont possibles :

- $f_i < r_i + p_i$ et $f_i < r_i' + p_i$ (*cf.* figure 9.2 (a)) : nous ajoutons dans les périodes de temps $\prec r_i + p_i + 1 \succ$, .., $\prec r_i' + p_i \succ$ la quantité a_{ik} dans l'histogramme des niveaux et l'activité i dans l'histogramme des activités;

- $f_i \geq r_i + p_i$ et $f_i < r_i' + p_i$ (*cf.* figure 9.2 (b)) : la quantité a_{ik} et l'activité i sont ajoutées respectivement dans l'histogramme des niveaux et dans l'histogramme des activités pendant les périodes de temps $\prec f_i + 1 \succ$, .., $\prec r_i' + p_i \succ$;

– $f_i \geq r_i + p_i$ et $f_i \geq r'_i + p_i$ (*cf.* figure 9.2 (c)) : dans ce cas, aucun des deux histogrammes n'est modifié.

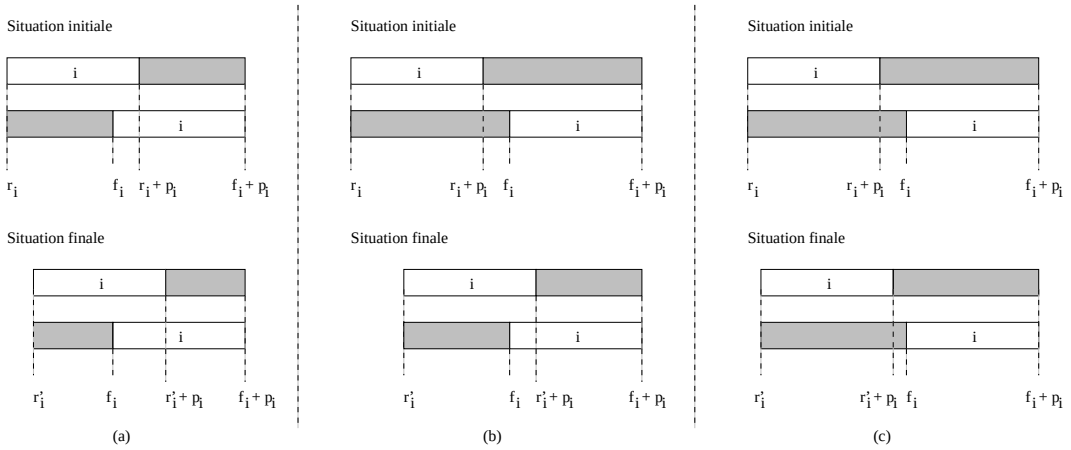


FIG. 9.2 – *Cas de l'augmentation de la borne inférieure d'une activité i*

9.2.3.2 La borne supérieure de la variable associée à une activité diminue

Dans le cas où l'événement rencontré lors de la propagation est la diminution de la date au plus tard de l'activité i de f_i à f'_i, alors nous avons les trois cas suivants :

– $f_i < r_i + p_i$ et $f'_i < r_i + p_i$ (*cf.* figure 9.3 (a)) : nous ajoutons dans les périodes de temps $\prec f'_i + 1 \succ$, .., $\prec f_i \succ$ la quantité a_{ik} dans l'histogramme des niveaux et l'activité i dans l'histogramme des activités;

– $f_i \geq r_i + p_i$ et $f'_i < r_i + p_i$ (*cf.* figure 9.3 (b)) : dans ce cas, nous ajoutons dans les périodes de temps $\prec f'_i + 1 \succ$, .., $\prec r_i + p_i \succ$ la quantité a_{ik} dans l'histogramme des niveaux et l'activité i dans l'histogramme des activités;

– $f_i \geq r_i + p_i$ et $f'_i \geq r_i + p_i$ (*cf.* figure 9.3 (c)) : les histogrammes ne sont pas modifiés.

9.2.3.3 La borne inférieure de la variable associée à une activité diminue

Si la date au plus tôt d'une activité i diminue de r_i à r'_i, trois cas sont possibles :

– $f_i < r_i + p_i$ et $f_i < r'_i + p_i$ (*cf.* figure 9.4 (a)) : nous retirons la quantité a_{ik} de l'histogramme des niveaux et l'activité i de l'histogramme des activités pendant les périodes de temps $\prec r'_i + p_i + 1 \succ$, .., $\prec r_i + p_i \succ$;

– $f_i < r_i + p_i$ et $f_i \geq r'_i + p_i$ (*cf.* figure 9.4 (b)) : la quantité a_{ik} et l'activité i sont alors retirés respectivement de l'histogramme des niveaux et de l'histogramme des activités pendant les périodes de temps $\prec f_i + 1 \succ$, .., $\prec r_i + p_i \succ$;

– $f_i \geq r_i + p_i$ et $f_i \geq r'_i + p_i$ (*cf.* figure 9.4 (c)) : dans ce cas, les histogrammes ne subissent aucun changement.

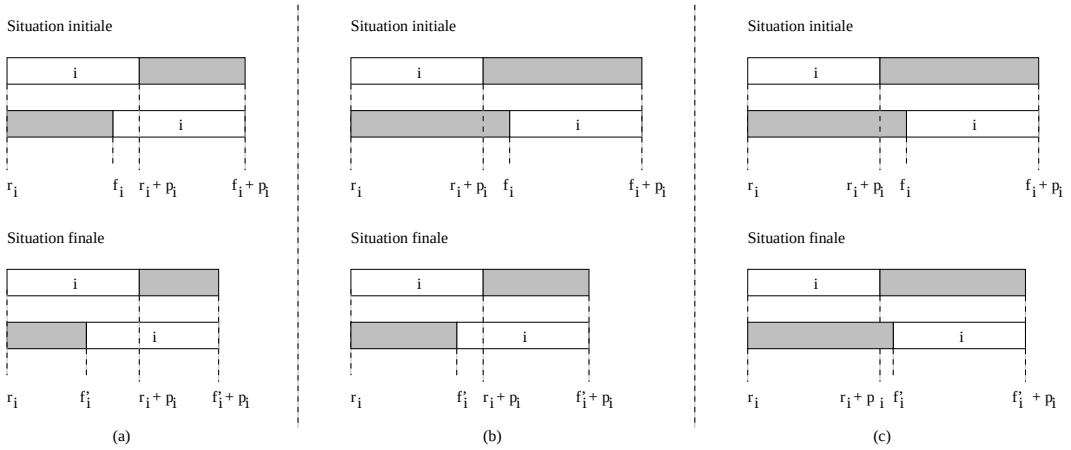


FIG. 9.3 – *Cas de la diminution de la borne supérieure d'une activité i*

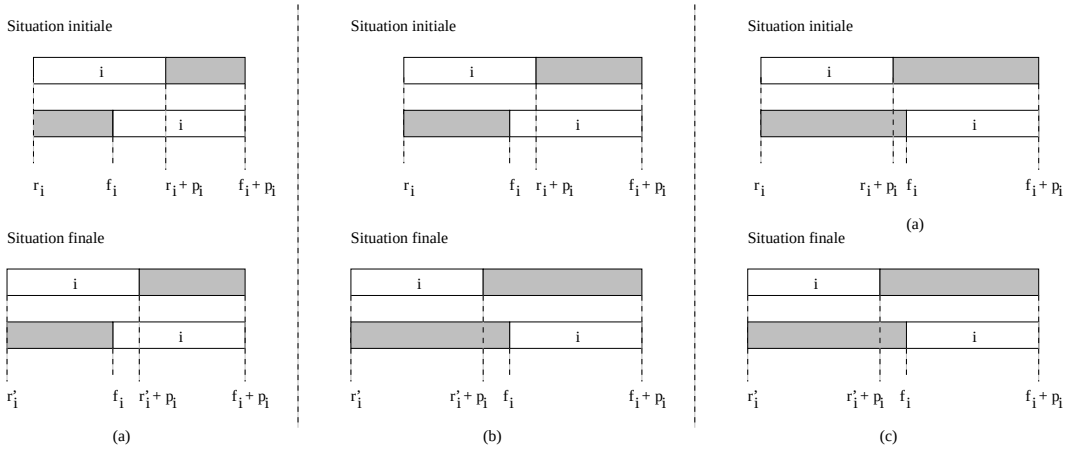


FIG. 9.4 – *Cas de la diminution de la borne inférieure d'une activité i*

9.2.3.4 La borne supérieure de la variable associée à une activité augmente

Lorsque l'événement rencontré au cours de la propagation est l'augmentation de la date au plus tard d'une activité i de f_i à $f_i^{'}$, alors les trois cas qui se présentent sont :

- $f_i < r_i + p_i$ et $f_i^{'} < r_i + p_i$ (*cf.* figure 9.5 (a)) : nous retirons la quantité a_{ik} de l'histogramme des niveaux et l'activité i de l'histogramme des activités durant les périodes de temps $\prec f_i + 1 \succ$, .., $\prec f_i^{'} \succ$;
- $f_i < r_i + p_i$ et $f_i^{'} \geq r_i + p_i$ (*cf.* figure 9.5 (b)) : nous retirons la quantité a_{ik} de l'histogramme des niveaux et l'activité i de l'histogramme des activités pendant les périodes de temps $\prec f_i + 1 \succ$, .., $\prec r_i + p_i \succ$;
- $f_i \geq r_i + p_i$ et $f_i^{'} \geq r_i + p_i$ (*cf.* figure 9.5 (c)) : les deux histogrammes ne sont pas modifiés.

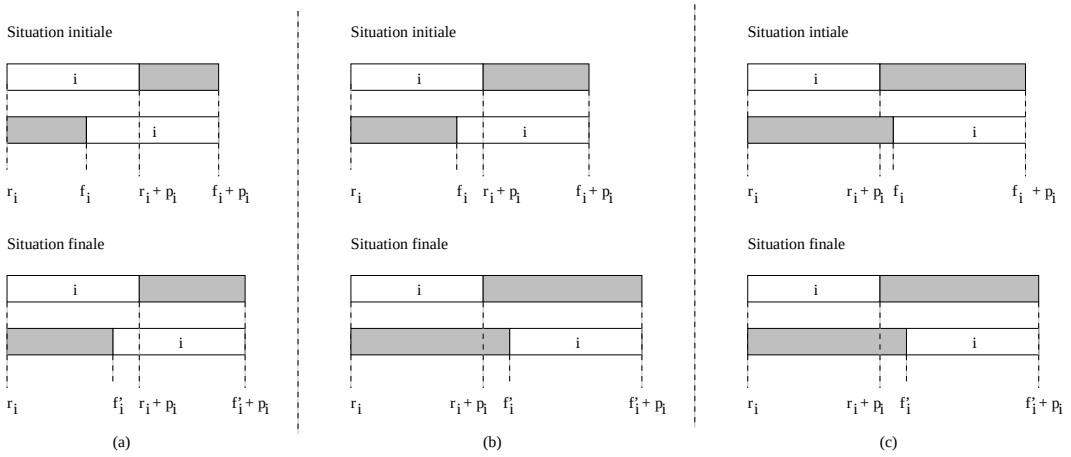


FIG. 9.5 – *Cas de l'augmentation de la borne supérieure d'une activité i*

9.2.4 Règles de propagation

Grâce à l'histogramme des niveaux nous pouvons vérifier, à chaque modification de l'une des bornes des domaines des activités, que le niveau de consommation ne dépasse pas la capacité limitée de la ressource. Dans le cas où cette capacité est dépassée, un conflit de ressource est alors détecté.

> **Exemple 25 (Détection d'un conflit) :**
> Dans l'exemple 24, si nous supposons que la capacité de la ressource est $R_1 = 7$ alors cette capacité est dépassée à la période $\prec 5 \succ$.

L'histogramme des niveaux peut aussi être utilisé pour ajuster les fenêtres temporelles des activités. Ainsi si, pour une période de temps $\prec t \succ$ incluse dans la fenêtre temporelle d'une activité i, la quantité de ressource restante n'est pas suffisante pour réaliser cette activité, alors les bornes de son domaine peuvent être ajustées.

Exemple 26 (Ajustements de la fenêtre temporelle d'une activité) :
Dans la figure 9.6, nous constatons que :
Pour le cas (a) : la ressource ne peut pas supporter l'activité i durant la période de temps $\prec 2 \succ$, et donc nous pouvons ajuster la date de début au plus tôt de l'activité i à la valeur 2.
Pour le cas (b) : l'activité i ne peut pas être exécutée durant la période de temps $\prec 8 \succ$, la date de fin au plus tard de l'activité i sera donc ajustée à la valeur 7.

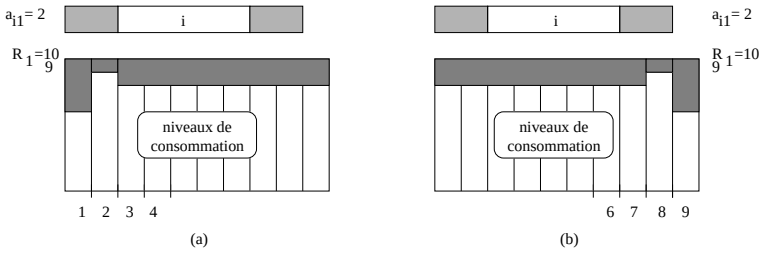


FIG. 9.6 – *Ajustement de la fenêtre temporelle d'une activité.*

L'algorithme de propagation que nous avons proposé ne vérifie le niveau de consommation que dans les périodes de temps où le niveau de consommation a augmenté. Dans le cas où aucun conflit n'est détecté, l'algorithme tente d'ajuster les bornes des fenêtres temporelles de l'activité considérée i. Nous signalons que le second test se fait uniquement sur une des parties de la fenêtre temporelle de l'activité i qui est en dehors de sa partie obligatoire $PO(i)$.

Considérons par exemple le cas où la date de début au plus tôt r_i d'une activité i a augmenté (le cas (a) de la figure 9.2). Les étapes suivies dans ce cas sont données dans l'algorithme 14. Dans cet algorithme, le second test ne sera fait que sur la partie gauche $[r_i^{'} + p_i + 1, f_i + p_i]$ de la fenêtre temporelle de l'activité i (*cf.* la ligne 1 de l'algorithme 14). Notons que, dans le cas où la date de début au plus tard f_i de l'activité i a diminué (le cas (b) de la figure 9.2), le second test sera fait uniquement sur la partie droite $[r_i^{'} + 1, f_i]$.

Nous avons choisi cette stratégie dans le but de réduire le nombre de tests. En effet :

- nous n'avons considéré que l'activité dont la borne inférieure a augmenté (ou la borne supérieure a diminuée). Sinon, nous risquerions de refaire les mêmes tests plusieurs fois.
- nous ne faisons le second test que sur une partie de la fenêtre temporelle de l'activité. Supposons par exemple que la borne inférieure d'une activité i a augmenté et que nous n'avons pas détecté de contradiction. Le second test est alors fait sur la partie gauche de la fenêtre temporelle de i. Dans le cas où ce test provoque un ajustement, la contrainte est réveillée de nouveau, et l'algorithme utilisé sera le symétrique de l'algorithme 14. Dans ce cas :
  - soit une contradiction est détéctée : nous avons donc évité de faire des tests sur la partie droite dans l'algorithme 14.
  - soit une contradiction n'est pas détectée : dans ce cas le test sera fait sur l'autre partie.

Algorithme 14: Règles de propagation : le cas (a) de la figure 9.2.

$\mathcal{C}_{po}(k)$ *est une contrainte parties obligatoires.*
i est l'activité dont la borne inférieure a augmentée de r_i *à* $r_i^{'}$.

Règles_Propagation($\mathcal{C}_{po}(k)$:**contrainte**, i:**Activité**, r_i:**entier**, $r_i^{'}$:**entier**)
début
  pour chaque $t \in [r_i + p_i + 1 \,..\, r_i^{'} + p_i]$ **faire**
    si $\mathcal{H}n_k[t] > R_k$ **alors**
      contradiction
  si *Pas de contradiction* **alors**
1   **pour chaque** $t \in [r_i^{'} + p_i + 1, f_i + p_i]$ **faire**
      si $\mathcal{H}n_k[t] + a_{ik} > R_k$ **alors**
        ajuster f_i *à* $t - p_i - 1$
fin

9.2.5 Calcul des explications

Après avoir présenté le principe de propagation de la contrainte *parties obligatoires*, nous allons nous intéresser dans cette partie au calcul des explications associées aux conflits de ressources et aux ajustements des fenêtres temporelles. Nous remarquons que seul l'histogramme des niveaux est utile pour détecter les conflits et réduire les fenêtres temporelles. L'histogramme des activités sera essentiellement utilisé pour le calcul des explications.

9.2.5.1 Conflit de ressource

Un conflit de ressource peut être détecté lorsque la borne inférieure d'une activité i augmente ou que sa borne supérieure diminue.

Si un conflit de ressource est détecté à une période de temps $\prec t \succ$. Ce conflit est dû :

- premièrement aux contraintes qui ont obligé les activités à appartenir à l'histogramme des activités $\mathcal{H}a_k$ à la période $\prec t \succ$. L'explication associée n'est autre que l'union des explications de la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j);$$

- et deuxièmement, à la contrainte de ressource $\mathcal{C}_{po}(k)$.

L'explication associée à ce conflit sera donc :

$$\left(\bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j) \right) \bigwedge \mathcal{C}_{po}(k)$$

Exemple 27 (Explication du conflit) :
Dans l'exemple 25, nous avons un conflit de ressource à la période $\prec 5 \succ$.
L'explication de ce conflit est :

$$\left(\wedge_{i\in\{1,2,3\}}\texttt{theDom}(t_i)\right) \wedge \mathcal{C}_{po}(1)$$

9.2.5.2 Ajustement d'une fenêtre temporelle

L'ajustement de la fenêtre temporelle d'une activité i ne peut être possible que lorsque la borne inférieure de son domaine augmente ou lorsque sa borne supérieure diminue.

Supposons qu'un ajustement de la fenêtre temporelle d'une activité i a été provoqué par l'augmentation de la borne inférieure de son domaine, et par l'augmentation de la quantité de ressource utilisée sur une période de temps $\prec t \succ$ dans l'histogramme des niveaux. Cet ajustement sera donc dû :

- premièrement, aux contraintes qui ont causé l'augmentation de la borne inférieure de l'activité i. L'explication associée est : $\texttt{theInf}(t_i)$;
- deuxièmement, aux contraintes qui ont imposé aux activités d'appartenir à l'histogramme des activités $\mathcal{H}a_k$ à la période $\prec t \succ$. L'explication associée n'est autre que l'union des explications de la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j\in\mathcal{H}a_k[t]} \texttt{theDom}(t_j);$$

- et enfin, à la contrainte de ressource $\mathcal{C}_{po}(k)$.

L'explication associée à cet ajustement sera donc :

$$\left(\bigwedge_{j\in\mathcal{H}a_k[t]} \texttt{theDom}(t_j)\right) \bigwedge \texttt{theInf}(t_i) \bigwedge \mathcal{C}_{po}(k)$$

Exemple 28 (Explication de l'ajustement de la fenêtre temporelle d'une activité) :
Reprenons l'exemple 26. Supposons que, dans le cas (a), la borne supérieure de l'activité i diminue. L'explication de l'ajustement de la date de début au plus tôt de l'activité i est :

$$\left(\wedge_{j\in\mathcal{H}a_k[2]}\texttt{theDom}(t_j)\right) \wedge \texttt{theSup}(t_i) \wedge \mathcal{C}_{po}(k)$$

Supposons que, dans le cas (b), la borne inférieure de l'activité i a augmentée. L'explication de l'ajustement de la date de début au plus tard de l'activité i est :

$$\left(\wedge_{j\in\mathcal{H}a_k[8]}\texttt{theDom}(t_j)\right) \wedge \texttt{theInf}(t_i) \wedge \mathcal{C}_{po}(k)$$

Algorithme 15: Règles de propagation avec explications : le cas (a) de la figure 9.2.

$\mathcal{C}_{po}(k)$ *est une contrainte parties obligatoires.*
i est l'activité dont la borne inférieure a augmentée de r_i *à* $r_i^{'}$.

$R\grave{e}gles\_Propagation(\mathcal{C}_{po}(k)$: **contrainte**, i: **Activité**, r_i: **entier**, $r_i^{'}$: **entier**)
début
  pour chaque $t \in [r_i + p_i + 1..r_i^{'} + p_i]$ **faire**
    si $\mathcal{H}n_k[t] > R_k$ **alors**
      Contradiction, l'explication de ce conflit est : $\bigwedge_{i \in \mathcal{H}a_k[t]} \texttt{theDom}(t_i) \bigwedge \mathcal{C}_{po}(k)$
  si *Pas de contradiction* **alors**
    pour chaque $t \in [r_i^{'} + p_i + 1, f_i + p_i]$ **faire**
      si $\mathcal{H}n_k[t] + a_{ik} > R_k$ **alors**
        $updateSup(t_i,\ t - p_i - 1, becauseOf(theConstraint(\mathcal{C}_{po}(k), \bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j) \bigwedge \texttt{theInf}(t_i))$
fin

Si nous reprenons l'algorithme 14, qui décrit les étapes suivies dans le cas (a) de la figure 9.2, sa version expliquée sera l'algorithme 15.

9.3 Contrainte basée sur la notion d'intervalles de tâches

La contrainte *parties obligatoires* que nous avons traitée dans la section précédente, est basée sur les fenêtres temporelles des activités. Cette contrainte ne détecte des conflits que lorsque l'histogramme des niveaux contient des sommets, et ne permet pas de repérer des conflits avant d'atteindre ces sommets. Pour cela, nous avons introduit une seconde contrainte basée sur la notion d'énergie, et plus particulièrement sur la notion d'intervalle de tâches [Caseau et Laburthe, 1996b].

9.3.1 Intervalles de tâches : rappels

Nous rappelons qu'un intervalle de tâches $IT = [i, j]$ associé à deux activités distinctes (ou non) i et j utilisant la même ressource k et satisfaisant les deux conditions : $r_i \leq r_j$ et $d_i \leq d_j$, est constitué de l'ensemble des activités l utilisant la même ressource k et vérifiant les deux conditions : $r_i \leq r_l$ et $d_l \leq d_j$.

Exemple 29 (Intervalle de tâches) :
Considérons un problème d'ordonnancement comportant six activités numérotées de 1 à 6, et une seule ressource. Nous associons à chaque activité i, sa date de début au plus tôt r_i, sa date de début au plus tard f_i, et sa quantité de ressource a_{i1}. Le tableau 9.2 fournit les données du problème.
La figure 9.7 présente l'intervalle de tâches associées aux deux activités 2 et 3. Nous avons : $IT = [2,3] = \{1,2,3,4\}$.

i	p_i	r_i	f_i	a_{i1}
1	4	2	4	2
2	5	1	2	1
3	5	2	4	2
4	3	3	5	3
5	3	0	3	3
6	4	6	8	3

TAB. 9.2 – *Données de l'exemple 29*

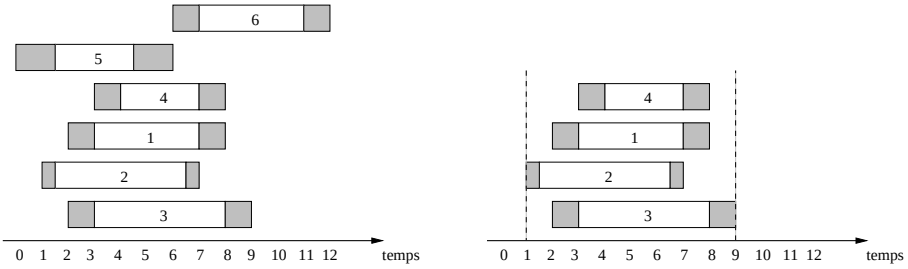


FIG. 9.7 – *Intervalle de tâches IT = [2,3].*

9.3.2 Modélisation

Dans notre modèle, nous associons à chaque intervalle de tâches $IT = [i, j]$, sur une ressource k, une contrainte $\mathcal{C}_{it}[IT, k]$ définie par :

- $début(\mathcal{C}_{it}[IT, k])$ et $fin(\mathcal{C}_{it}[IT, k])$ qui correspondent respectivement aux deux activités i et j;
- $intérieur(\mathcal{C}_{it}[IT, k])$ constitué de l'ensemble des activités qui sont à l'intérieur de l'intervalle de tâches IT (y compris les deux activités i et j);
- $extérieur(\mathcal{C}_{it}[IT, k])$ qui contient l'ensemble des activités qui sont à l'extérieur de l'intervalle de tâches IT. En d'autre terme, ce sont les activités n'appartenant pas à l'ensemble $intérieur(\mathcal{C}_{it}[IT, k])$;
- $énergie(\mathcal{C}_{it}[IT, k])$ représentant la quantité d'énergie consommée à l'intérieur de l'intervalle de tâches IT. Elle est égale à la somme des énergies $w_{lk} = p_l \times a_{lk}$ de toutes les activités l appartenant à $intérieur(\mathcal{C}_{it}[IT, k])$. Ainsi, nous avons :

$$énergie(\mathcal{C}_{it}[IT, k]) = \sum_{l \in intérieur(\mathcal{C}_{it}[IT,k])} w_{lk}$$

Exemple 30 (Contrainte intervalle de tâches) :
La contrainte $\mathcal{C}_{it}[IT, 1]$ associée à l'intervalle de tâches $IT = [2, 3]$ de l'exemple 29 est définie par :
- $début(\mathcal{C}_{it}[IT, 1]) = 2$ et $fin(\mathcal{C}_{it}[IT, 1]) = 3$
- $intérieur(\mathcal{C}_{it}[IT, 1]) = \{1, 2, 3, 4\}$
- $extérieur(\mathcal{C}_{it}[IT, 1]) = \{5, 6\}$
- $énergie(\mathcal{C}_{it}[IT, 1]) = \sum_{i \in \{1,2,3,4\}} w_{i1} = 32$

9.3.3 Maintien des structures de données

Nous avons modélisé les contraintes *intervalles de tâches* comme des contraintes globales. Nous associons à chaque contrainte $\mathcal{C}_{it}[IT,k]$ l'ensemble des variables associées aux activités utilisant la ressource k. Le mécanisme de propagation de ces contraintes est basé sur la notion d'événements. Les événements qui peuvent être rencontrés au cours de la propagation sont l'augmentation ou la diminution de la borne inférieure ou de la borne supérieure de la variable associée à une des activités.

Après chaque réduction ou élargissement de la fenêtre temporelle de l'une des activités, les caractéristiques des contraintes *intervalles de tâches* doivent être mises à jour. Le système permettant de gérer la propagation des événements est basé sur les règles suivantes :

9.3.3.1 La borne inférieure du domaine de la variable associée à une activité augmente

Dans le cas où la borne inférieure d'une activité augmente, les contraintes *intervalles de tâches* qui sont concernées sont celles dont l'activité est égale à $début(\mathcal{C}_{it}[IT,k])$, et celles dont l'activité est un élément de $extérieur(\mathcal{C}_{it}[IT,k])$.

Cas où la borne inférieure du domaine de la variable associée à une activité $début(\mathcal{C}_{it}[IT,k])$ augmente : dans ce cas, il est possible que certaines activités de l'ensemble $intérieur(\mathcal{C}_{it}[IT,k])$ deviennent des éléments de l'ensemble $extérieur(\mathcal{C}_{it}[IT,k])$. C'est le cas de l'activité l dans la figure (*cf.* figure 9.8 (a)). Nous déroulons alors l'algorithme 16.

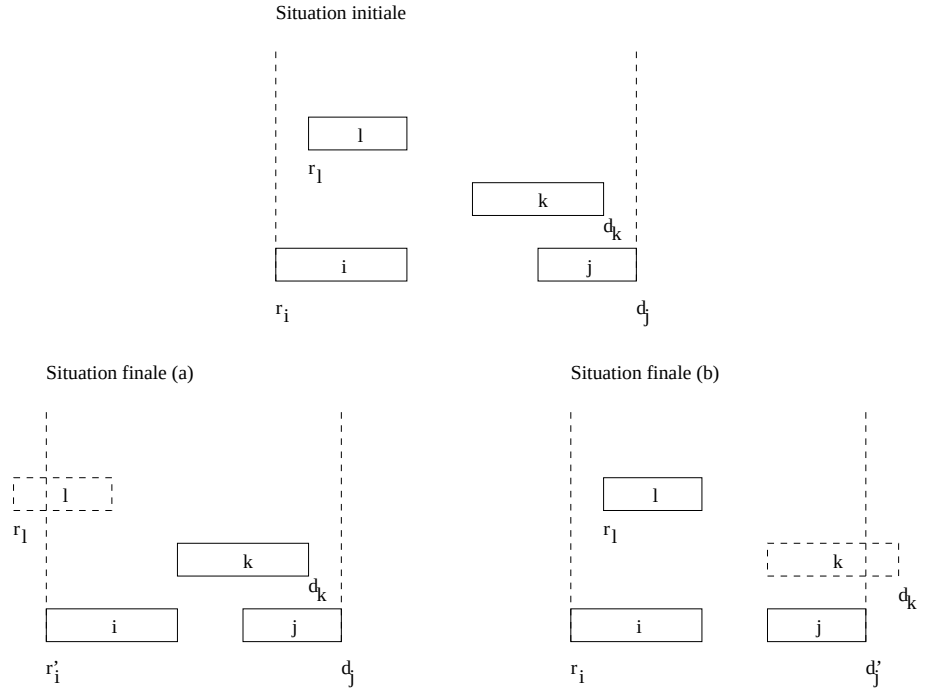


FIG. 9.8 – *(a) représente le cas de l'augmentation de la borne inférieure d'une activité constituant le début de l'intervalle de tâches. (b) représente le cas symétrique.*

Algorithme 16: Maintien des structures de données dans le cas (a) de la figure 9.8.

$\mathcal{C}_{it}[IT,k]$ *est une contrainte intervalles de tâches de début*$(\mathcal{C}_{it}[IT,k]) = i$.
i *est l'activité dont la borne inférieure a augmentée de* r_i *à* $r_i^{'}$.

Maintien_Structures_Données($\mathcal{C}_{it}[IT,k]$:**contrainte**, i:**Activité**, r_i:**entier**, $r_i^{'}$:**entier**)

début
  pour chaque $l \in$ *intérieur*$(\mathcal{C}_{it}[IT,k])$ **faire**
    si $r_l < r_i^{'}$ **alors**
      intérieur$(\mathcal{C}_{it}[IT,k]) \leftarrow$ *intérieur*$(\mathcal{C}_{it}[IT,k]) \setminus \{l\}$
      extérieur$(\mathcal{C}_{it}[IT,k]) \leftarrow$ *extérieur*$(\mathcal{C}_{it}[IT,k]) \cup \{l\}$
      énergie$(\mathcal{C}_{it}[IT,k]) \leftarrow$ *énergie*$(\mathcal{C}_{it}[IT,k]) - w_{lk}$
fin

Algorithme 17: Maintien des structures de données dans le cas (a) de la figure 9.9.

l *est l'activité dont la borne inférieure augmente de* r_l *à* $r_l^{'}$.
$\mathcal{C}_{it}[IT,k]$ *est une contrainte intervalles de tâches tels que* $l \in$ *extérieur*$(\mathcal{C}_{it}[IT,k])$.
i *est l'activité début*$(\mathcal{C}_{it}[IT,k])$.

Maintien_Structures_Données($\mathcal{C}_{it}[IT,k]$:**contrainte**, l:**Activité**, r_l:**entier**, $r_l^{'}$:**entier**)

début
  si $r_l^{'} \geq r_i$ *et* $f_l + p_l \leq d_j$ **alors**
    intérieur$(\mathcal{C}_{it}[IT,k]) \leftarrow$ *intérieur*$(\mathcal{C}_{it}[IT,k]) \cup \{l\}$
    extérieur$(\mathcal{C}_{it}[IT,k]) \leftarrow$ *extérieur*$(\mathcal{C}_{it}[IT,k]) \setminus \{l\}$
    énergie$(\mathcal{C}_{it}[IT,k]) \leftarrow$ *énergie*$(\mathcal{C}_{it}[IT,k]) + w_{lk}$
fin

Cas où l'activité l dont la borne inférieure augmente est un élément de *extérieur*$(\mathcal{C}_{it}[IT,k])$ **:** dans ce cas, l'activité l peut devenir un élément de l'ensemble *intérieur*$(\mathcal{C}_{it}[IT,k])$ (*cf.* figure 9.9 (a)). La mise à jour des caractéristiques de cette contrainte se fait grâce à l'algorithme 17.

9.3.3.2 La borne supérieure du domaine de la variable associée à une activité diminue

Nous pouvons remarquer que ce cas est symétrique au précédent (*cf.* figure 9.8 (b), et figure 9.9 (b)). Les contraintes concernées sont les contraintes dont l'activité est égale à $fin(\mathcal{C}_{it}[IT,k])$, et celles dont l'activité est un élément de l'ensemble *extérieur*$(\mathcal{C}_{it}[IT,k])$.

9.3.3.3 La borne inférieure du domaine de la variable associée à une activité diminue

Lorsque la borne inférieure du domaine d'une activité diminue, nous mettons à jour les caractéristiques des contraintes dont l'activité est égale à *début*$(\mathcal{C}_{it}[IT,k])$, et celles dont l'activité fait partie de l'ensemble *extérieur*$(\mathcal{C}_{it}[IT,k])$.

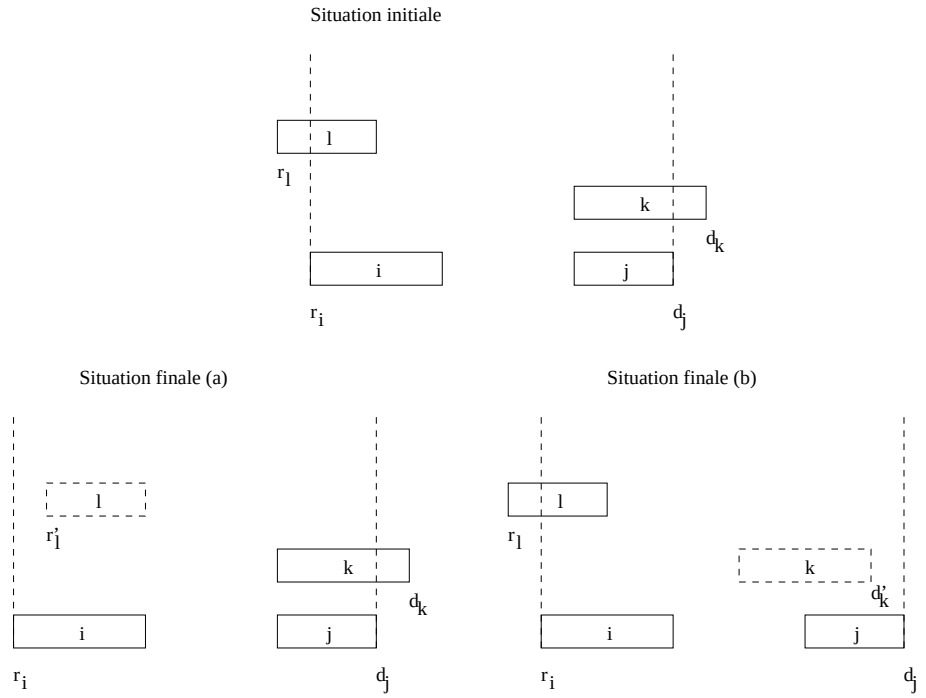


FIG. 9.9 – *(a) représente le cas de l'augmentation de la borne inférieure d'une activité extérieure à l'intervalle de tâches. (b) représente le cas symétrique.*

Cas où la borne inférieure de la variable associée à une activité $début(\mathcal{C}_{it}[IT,k])$ **diminue :** certaines activités de $extérieur(\mathcal{C}_{it}[IT,k])$ peuvent devenir des éléments de $intérieur(\mathcal{C}_{it}[IT,k])$ (cas de l'activité l dans la figure 9.10 (a)). Nous déroulons alors l'algorithme 18.

Cas où l'activité l **dont la borne inférieure diminue est un élément de** $intérieur(\mathcal{C}_{it}[IT,k])$ **:** l'activité l peut sortir de $intérieur(\mathcal{C}_{it}[IT,k])$ et devenir un élément de $extérieur(\mathcal{C}_{it}[IT,k])$ (*cf.* figure 9.11 (a)). La mise à jour se fait par l'algorithme 19.

9.3.3.4 La borne supérieure du domaine de la variable associée à une activité augmente

Par symétrie, comme le montre les deux figures : 9.10 (b) et 9.11 (b), nous mettons à jour les caractéristiques des contraintes dont l'activité concernée est égale à $fin(\mathcal{C}_{it}[IT,1])$, et celles dont l'activité est un élément de l'ensemble $intérieur(\mathcal{C}_{it}[IT,1])$.

9.3.4 Règles de propagation

Parmi les règles de déduction introduites par *Caseau et Laburthe* [1994], les seules règles qui utilisent la notion d'énergie sont : la *règle d'intégrité* (*integrity rule*) et la *règle du jet* (*Throw rule*). Ces deux règles ont pour but de détecter des conflits de ressource et d'ajuster les fenêtres temporelles des activités. Nous avons utilisé et amélioré ces deux règles en prenant

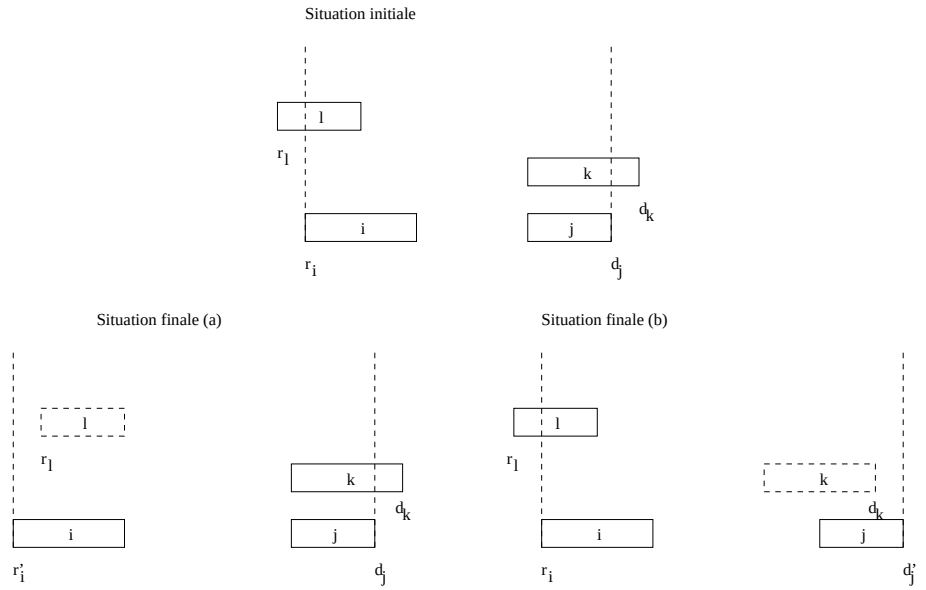


FIG. 9.10 – *(a) représente le cas où la borne inférieure de l'activité constituant le début de l'intervalle de tâches a diminué. Le cas (b) représente le cas symétrique.*

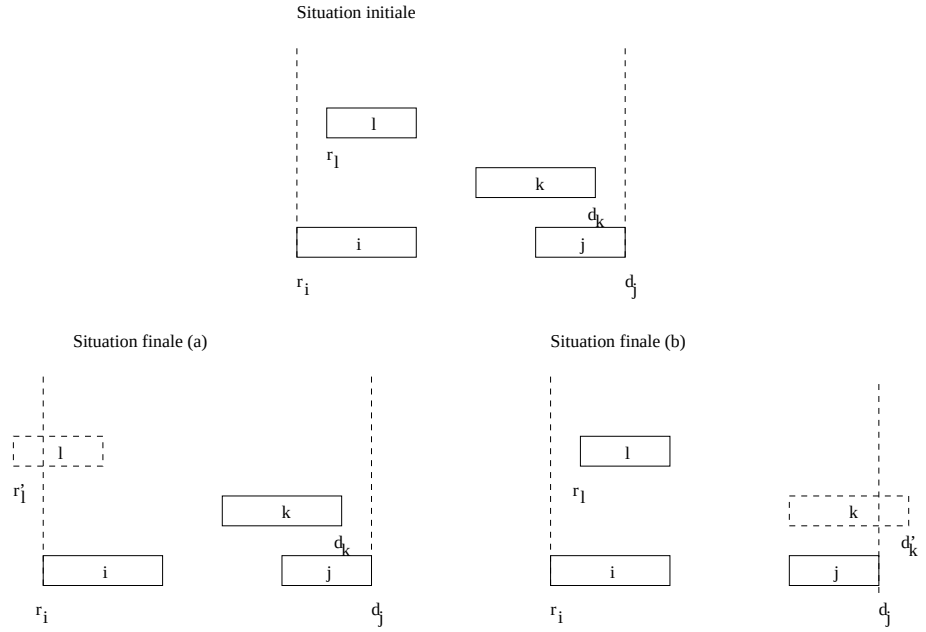


FIG. 9.11 – *(a) représente le cas où la borne supérieure d'une activité de l'extérieur de l'intervalle de tâches a augmenté. Le cas (b) représente le cas symétrique.*

Algorithme 18: Maintien des structures de données dans le cas (a) de la figure 9.10.

$\mathcal{C}_{it}[IT,k]$ *est une contrainte intervalles de tâches tels que début*$(\mathcal{C}_{it}[IT,k]) = i$.
i *est l'activité dont la borne inférieure diminue de* r_i *à* $r_i^{'}$.

$Maintien\_Structures\_Données$($\mathcal{C}_{it}[IT,k]$:**contrainte**, i:**Activité**, r_i:**entier**, $r_i^{'}$:**entier**)

début
  pour chaque $l \in extérieur(\mathcal{C}_{it}[IT,k])$ **faire**
    si $r_l \geq r_i^{'}$ *et* $f_l + p_l \leq d_j$ **alors**
      $intérieur(\mathcal{C}_{it}[IT,k]) \leftarrow intérieur(\mathcal{C}_{it}[IT,k]) \cup \{l\}$
      $extérieur(\mathcal{C}_{it}[IT,k]) \leftarrow extérieur(\mathcal{C}_{it}[IT,k]) \setminus \{l\}$
      $énergie(\mathcal{C}_{it}[IT,k]) \leftarrow énergie(\mathcal{C}_{it}[IT,k]) + w_{lk}$
fin

Algorithme 19: Maintien des structures de données dans le cas (a) de la figure 9.11.

l *est l'activité dont la borne inférieure diminue de* r_l *à* $r_l^{'}$.
$\mathcal{C}_{it}[IT,k]$ *est une contrainte intervalles de tâches telle que* $l \in intérieur(\mathcal{C}_{it}[IT,k])$.
i *est l'activité début*$(\mathcal{C}_{it}[IT,k])$.

$Maintien\_Structures\_Données$($\mathcal{C}_{it}[IT,k]$:**contrainte**, l:**Activité**, r_l:**entier**, $r_l^{'}$:**entier**)

début
  si $r_l^{'} < r_i$ **alors**
    $intérieur(\mathcal{C}_{it}[IT,k]) \leftarrow intérieur(\mathcal{C}_{it}[IT,k]) \setminus \{l\}$
    $extérieur(\mathcal{C}_{it}[IT,k]) \leftarrow extérieur(\mathcal{C}_{it}[IT,k]) \cup \{l\}$
    $énergie(\mathcal{C}_{it}[IT,k]) \leftarrow énergie(\mathcal{C}_{it}[IT,k]) - w_{lk}$
fin

en compte les activités qui intersectent les *intervalles de tâches*. Tout d'abord, nous allons rappeler ces deux règles.

9.3.4.1 Règle d'integrité

Cette règle détecte un conflit de ressource lorsque l'énergie associée à la contrainte $\mathcal{C}_{it}[IT,k]$ dépasse la quantité d'énergie disponible sur l'intervalle de tâches IT, cette quantité d'énergie disponible étant égale à $énergie(IT,k) = (d_j - r_i) \times R_k$.

$$Si\ énergie(\mathcal{C}_{it}[IT,k]) > énergie(IT,k)\ alors\ nous\ avons\ un\ conflit$$

Exemple 31 (Règle d'intégrité) :
Reprenons l'exemple 29, et supposons que la capacité de la ressource est $R_1 = 3$.
En appliquant la régle d'intégrité, nous détectons un conflit de ressource. En effet, $énergie(\mathcal{C}_{it}[IT,1]) = 32 > énergie(IT,1) = 8 \times 3$.

9.3.4.2 Règle de jet

Cette règle considère un intervalle de tâches $IT = [i, j]$, et une activité l qui chevauche cet intervalle, *i.e.* telle que $[r_l, d_l] \cap [r_i, d_j] \neq \emptyset$. Si la quantité de ressource disponible sur l'intervalle IT n'est pas suffisante pour exécuter les activités de IT et la partie de l qui le chevauche lorsque l est calée à gauche (*resp.* à droite), alors r_l (*resp.* f_l) doit être ajusté.

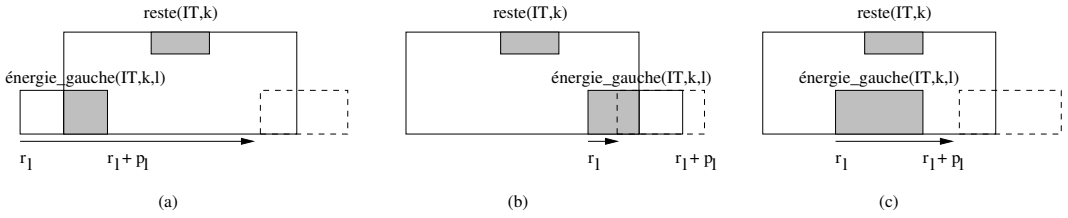


FIG. 9.12 – *Règle de jet lorsque l'activité est calée à gauche*

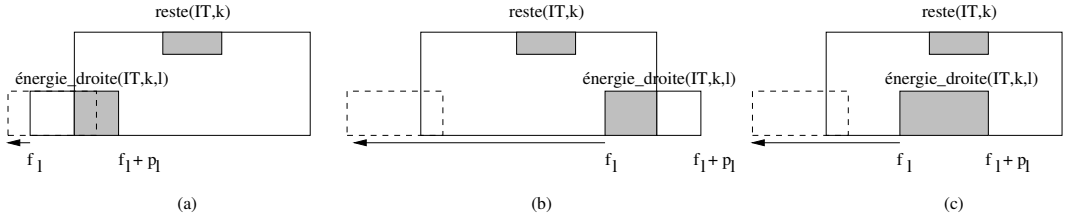


FIG. 9.13 – *Règle de jet lorsque l'activité est calée à droite*

Dans le cas où l'activité l est calée à gauche (*cf.* figure 9.12), nous pouvons calculer la quantité d'énergie $énergie\_gauche(IT, k, l)$ intersectant l'intervalle de tâches IT grâce à la formule :

$$énergie\_gauche(IT, k, l) = min\{w_{lk} - (r_i - r_l) \times a_{lk}, (f_j + p_j - r_l) \times a_{lk}, w_{lk}\}$$

Notons $reste(IT, k) = énergie(IT, k) - énergie(\mathcal{C}_{it}[IT, k])$ la quantité d'énergie restante sur l'intervalle IT. Nous avons alors l'ajustement suivant :

$$Si\ reste(IT, k) < énergie\_gauche(IT, k, l)\ alors\ r_l\ :=\ max\{r_l, f_j + p_j - \lfloor \frac{reste(IT, k)}{a_{lk}} \rfloor\}$$

Par symétrie, lorsque l'activité l est calée à droite (*cf.* figure 9.13), la quantité $énergie\_droite(IT, k, l)$ intersectant l'intervalle de tâches IT peut être calculée par la formule :

$$énergie\_droite(IT, k, l) = min\{(f_l + p_l - r_i) \times a_{lk}, w_{lk} - (f_l + p_l - f_j - p_j) \times a_{lk}, w_{lk}\}$$

Et nous avons l'ajustement suivant :

$$Si\ reste(IT, k) < énergie\_droite(IT, k, l)\ alors\ f_l\ :=\ min\{f_l, r_i - p_l + \lfloor \frac{reste(IT, k)}{a_{lk}} \rfloor\}$$

9.3.4.3 Règle d'integrité améliorée

Dans le cas général, nous avons trois cas possibles pour une activité appartenant à $extérieur(\mathcal{C}_{it}[IT,k])$ chevauchant l'intervalle de tâches IT :

- l'activité peut couvrir complètement l'intervalle de tâches IT quelque soit sa date de début dans son domaine : l'énergie réservée est $(d_j - r_i) \times a_{lk}$;
- la durée minimale de chevauchement de l'activité avec IT est obtenue en la faisant débuter le plus tôt possible : l'énergie réservée est $(r_l + p_l - r_i) \times a_{lk}$;
- la durée minimale de chevauchement de l'activité avec IT est obtenue en la faisant démarrer le plus tard possible : l'énergie réservée est $(d_j - f_l) \times a_{lk}$.

Donc, la quantité d'énergie minimale réservée par une activité l de $extérieur(\mathcal{C}_{it}[IT,k])$ peut être calculée par la formule :

$$énergie(IT,k,l) \;=\; a_{lk} \times \max\{0, \min\{d_j - r_i, r_l + p_l - r_i, d_j - f_l\}\}$$

Si nous notons $intersecte(\mathcal{C}_{it}[IT,k])$ l'ensemble des activités de $extérieur(\mathcal{C}_{it}[IT,k])$ qui intersectent l'intervalle de tâches IT (*i.e.*, $énergie(IT,k,l) \neq 0$), la règle d'integrité améliorée sera donc :

$$Si\ énergie(IT,k) \;+ \sum_{l \in intersecte(\mathcal{C}_{it}[IT,k])} énergie(IT,k,l) \;>\; (d_j - r_i) \times R_k$$

alors nous avons un conflit de ressource

Afin d'illustrer notre raisonnement, nous allons considérer l'exemple suivant :

Exemple 32 (Règle d'integrité améliorée) :
Reprenons l'exemple 29, et supposons que la capacité de la ressource est $R_1 = 5$.
En appliquant la régle d'intégrité, nous ne détectons pas de conflit de ressource. En effet, $énergie(\mathcal{C}_{it}[IT,1]) = 32 \leq énergie(IT,1) = 8 \times 5$.
Lorsque nous considérons une des parties (de l'activité 5 ou 6) intersectant l'intervalle de tâches IT, nous ne détectons pas de conflit. Car nous avons : $énergie(IT,1) + énergie(IT,1,5) = 32 + 6 \leq 40$, *et* $énergie(IT,1) + énergie(IT,1,6) = 32 + 3 \leq 40$.
Par contre, en considérant les deux parties qui intersectent l'intervalle de tâches $IT = [2,3]$, nous avons un conflict de ressource. En effet, $énergie(IT,1) + énergie(IT,1,5) + énergie(IT,1,6) = 32 + 6 + 3 > 40$.

9.3.4.4 Règle de jet améliorée

De la même façon que pour la règle précédente, la règle de jet peut être améliorée en prenant en compte l'énergie minimale consommée par les activités qui chevauchent l'intervalle IT.

Dans ce cas, la quantité d'énergie restante $reste(IT,k)$ devient :

$$reste'(IT,k,l) \ = \ (d_j - r_i) \times R_k \ - \ énergie(IT,k) \ - \sum_{l' \in intesecte(\mathcal{C}_{it}[IT,k]) \setminus \{l\}} énergie(IT,k,l')$$

Ainsi dans le cas où l'activité l est calée à gauche, nous avons l'ajustement :

$$Si\ reste'(IT,k) < énergie\_gauche(IT,k,l)\ alors\ r_l \ := \ max\{r_l, d_j - \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor\}$$

Et dans le cas où l'activité l est calée à droite, nous avons l'ajustement :

$$Si\ reste'(IT,k) < énergie\_droite(IT,k,l)\ alors\ f_l \ := \ min\{f_l, r_i - p_l + \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor\}$$

Dans le but d'illustrer cette règle, nous considérons l'exemple suivant :

> **Exemple 33 (Règle de jet améliorée) :**
> Reprenons l'exemple 29, et supposons que : $R_1 = 5$ et $a_{61} = 2$.
> Si nous appliquons la règle de jet à l'activité 6, en la calant à gauche, alors nous avons : $reste(IT,k) = 40 - 32 = 8 \geq énergie\_gauche(IT,1,6) = 6$, et par suite nous n'avons pas d'ajustement.
> Par contre, lorsque nous considérons en plus l'énergie minimale consommée par l'activité 5 qui chevauche l'intervalle IT, nous avons : $reste'(IT,k) = reste(IT,k) - énergie(IT,1,5) = 40 - 32 - 6 = 2$, ce qui est inférieur à la valeur de $énergie\_gauche(IT,1,6)$ qui vaut 6. Nous avons alors l'ajustement $r_6 := 8$.

9.3.4.5 Algorithme de propagation

L'algorithme de propagation (algorithme 20) regroupe les trois règles suivantes dans le cas où la borne inférieure du domaine d'une activité augmente :

- la règle d'intégrité (testée dans la ligne 1 de l'algorithme);
- la règle d'intégrité améliorée (ligne 2 dans l'algorithme). Cette règle ne sera testée que si la premiére n'a pas détecté de conflit;
- et la règle de jet améliorée (ligne 3 dans l'algorithme). Cette règle n'est appliquée que lorsqu'aucune des deux premières règles n'a détecté de conflit.

Comme tout les conflits détectés par la règle d'intégrité sont aussi détectés par la règle d'intégrité améliorée, nous pouvons nous contenter de ne tester que la seconde règle. Mais, comme nous avons besoin d'avoir des explications fines et que les explications associées à la première règle sont plus fines que celles fournies par la seconde (comme nous le verrons par la suite), nous avons choisi de n'utiliser la deuxième que si la première n'a aucun effet.

Aussi, nous remarquons que tout ajustement provoqué par la règle de jet est aussi provoqué par la règle de jet améliorée. Dans notre algorithme, nous n'avons conservé que la version améliorée car les ajustements qu'elle calcule sont plus précis.

Algorithme 20: Règles de propagation : cas où la borne inférieure du domaine d'une activité augmente.

l est l'activité dont la borne inférieure augmente de r_l à r_l'.
$\mathcal{C}_{it}[IT,k]$ est une contrainte intervalles de tâches tels que $l \in extérieur(\mathcal{C}_{it}[IT,k])$.

Règle_Propagation($\mathcal{C}_{it}[IT,k]$:**contrainte**, l:**Activité**, r_l:**entier**, r_l':**entier**)
début
1 **si** $reste(IT,k) < 0$ **alors**
contradiction
sinon
2 **si** $énergie(IT,k) + \sum_{l \in intersecte(\mathcal{C}_{it}[IT,k])} énergie(IT,k,l) > (d_j - r_i) \times R_k$ **alors**
contradiction
sinon
3 **si** $reste'(IT,k) < énergie\_gauche(IT,k,l)$ **alors**
$ajuster\ r_l\ à\ (d_j - \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor)$
fin

L'algorithme de propagation appliqué dans les cas d'une diminution de la borne supérieure du domaine d'une activité peut facilement être déduit de celui-ci. Par contre, dans le cas où la borne inférieure du domaine d'une activité diminue, ou la borne supérieure du domaine d'une activité augmente nous n'appliquons aucun algorithme de propagation.

9.3.5 Calcul des explications

Dans la partie précédente, nous avons présenté un mécanisme de propagation des contraintes *intervalles de tâches* basé sur trois règles : la règle d'intégrité, la règle d'intégrité améliorée, et la règle du jet améliorée. Dans cette partie, nous nous intéressons au calcul des explications associées à chacune de ces règles.

La règle d'intégrité : Cette règle localise un conflit de ressource lorsque la quantité d'énergie nécessaire pour exécuter toutes les activités de $intérieur(\mathcal{C}_{it}[IT,k])$ dépasse l'énergie disponible sur l'intervalle de tâches IT. Un tel conflit sera dû :

- premièrement, aux contraintes qui ont forcé les activités à devenir des éléments de $intérieur(\mathcal{C}_{it}[IT,k])$. L'explication associée n'est autre que l'union des explications qui ont causé la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j);$$

- et deuxièmement, à la contrainte de ressource $\mathcal{C}_{it}[IT,k]$.

L'explication associée à ce conflit sera donc :

$$\left(\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \right) \bigwedge \mathcal{C}_{it}[IT,k]$$

Exemple 34 (Règle d'intégrité avec explication) :
Reprenons l'exemple 31.
En appliquant la régle d'intégrité, nous détectons un conflit de ressource. L'explication de ce conflit est :

$$\left(\wedge_{i\in\{1,2,3,4\}}\texttt{theDom}(t_i)\right) \wedge \mathcal{C}_{it}[IT,1]$$

La règle d'intégrité améliorée : Cette règle améliorée prend en compte les activités de $extérieur(\mathcal{C}_{it}[IT,k])$ qui intersectent l'intervalle de tâches IT. Si un conflit est détecté, il est dû :

- premièrement, aux contraintes qui ont forcé les activités à devenir des éléments de $intérieur(\mathcal{C}_{it}[IT,k])$. L'explication associée n'est autre que l'union des explications de la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j\in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j);$$

- deuxiémement, aux contraintes qui ont obligé les activités de $intersecte(\mathcal{C}_{it}[IT,k])$ à intersecter l'intervalle de tâches IT. L'explication associée sera donc l'union des explications de la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j\in intersecte(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j);$$

- et enfin, à la contrainte *intervalle de tâches* $\mathcal{C}_{it}[IT,k]$.

L'explication associée à ce conflit est donc :

$$\bigwedge_{j\in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \bigwedge_{j\in intersecte(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \bigwedge \mathcal{C}_{it}[IT,k]$$

Exemple 35 (Règle d'integrité améliorée avec explication) :
Reprenons l'exemple 32.
En appliquant la régle d'intégrité améliorée, et en considérant les deux parties qui intersectent l'intervalle de tâches $IT = [2,3]$, nous avons un conflict de ressource. L'explication de ce conflit est :

$$\bigwedge_{j\in\{1,2,3,4\}} \texttt{theDom}(t_j) \bigwedge_{j\in\{5,6\}} \texttt{theDom}(t_j) \bigwedge \mathcal{C}_{it}[IT,1]$$

La règle du jet améliorée : L'ajustement de la fenêtre temporelle d'une activité i ne sera possible que s'il n'y a pas assez d'énergie pour l'exécuter à sa date de début au plus tôt r_i ou à sa date de début au plus tard f_i.

Supposons que la date de début au plus tôt r_l d'une activité l ait été ajustée à $r_l^{'}$ (grâce à la règle du jet améliorée). Cet ajustement a été provoqué à cause de l'augmentation de la valeur actuelle de la borne inférieure de l'activité l, et à l'insuffisance d'énergie pour pouvoir commencer l'activité l à l'instant $r_l^{'}$. Cet ajustement sera donc dû :

- premièrement, aux contraintes qui ont permis à de la borne inférieure de l'activité l d'atteindre la valeur r_l. L'explication associée est : $\texttt{theInf}(t_l)$;
- deuxièmement, aux contraintes qui ont imposé aux activités de devenir des éléments de $intérieur(\mathcal{C}_{it}[IT,k])$. L'explication associée est l'union des explications qui ont causé la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j);$$

- troisièmement, aux contraintes qui ont obligé les activités de $intersecte(\mathcal{C}_{it}[IT,k])$ (sauf l'activité i) à intersecter l'intervalle de tâches IT. L'explication associée est l'union des explications qui ont causé la réduction des domaines de chacune de ces activités :

$$\bigwedge_{j \in intersecte(\mathcal{C}_{it}[IT,k]) \setminus \{i\}} \texttt{theDom}(t_j);$$

- et enfin, à la contrainte de ressource $\mathcal{C}_{it}[IT,k]$.

L'explication associée à cet ajustement est donc :

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}_{it}[IT,k]) \setminus \{i\}} \texttt{theDom}(t_j) \bigwedge \texttt{theInf}(t_l) \bigwedge \mathcal{C}_{it}[IT,k]$$

Exemple 36 (Règle de jet améliorée avec explication) :
Reprenons l'exemple 33.
Lorsque nous appliquons la règle de jet amélioree à l'activité 6, en la calant à gauche, nous avons l'ajustement $r_6 := 8$.
L'explication de cet ajustement est :

$\wedge_{j \in \in \{1,2,3,4\}} \texttt{theDom}(t_j) \wedge_{j \in \{5,6\} \setminus \{6\}} \texttt{theDom}(t_j) \wedge \texttt{theInf}(t_6) \wedge \mathcal{C}_{it}[IT,1]$

La version expliquée de l'algorithme 20 décrivant les étapes suivit pour propager les contraintes *intervalles de tâches* est donné par l'algorithme 21.

9.4 Conclusion

Dans ce chapitre, nous avons développé deux types de contraintes de ressources permettant de détecter des conflits et d'ajuster les fenêtres temporelles des activités. Ces contraintes sont

Algorithme 21: Règles de propagation avec explications.

l est l'activité dont la borne inférieure a augmenté de r_l à r_l'.
$\mathcal{C}_{it}[IT,k]$ est une contrainte intervalles de tâches telque $l \in extérieur(\mathcal{C}_{it}[IT,k])$.

Règle_Propagation($\mathcal{C}_{it}[IT,k]$:**contrainte**, l:**Activité**, r_l:**entier**, r_l':**entier**)
début
  si $reste(IT,k) < 0$ **alors**
    Contradiction, l'explication de ce conflit est :

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \bigwedge \mathcal{C}_{it}[IT,k]$$

  sinon
    si $énergie(IT,k) + \sum_{l \in extérieur(\mathcal{C}_{it}[IT,k])} énergie(IT,k,l) > (f_j + p_j - r_i) \times R_k$ **alors**
      Contradiction, l'explication de ce conflit :

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \bigwedge \mathcal{C}_{it}[IT,k]$$

    sinon
      si $reste'(IT,k) < énergie\_gauche(IT,k,l)$ **alors**
        $updateInf(t_l, d_j - \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor, becauseOf(thConstraint(\mathcal{C}_{it}[IT,k]),$

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}_{it}[IT,k]) \setminus \{i\}} \texttt{theDom}(t_j) \bigwedge \texttt{theInf}(t_l))$$

fin

essentiellement basées sur des règles de déductions que nous avons améliorées pour détecter plus de conflits et avoir des réductions plus fines des fenêtres temporelles.

Le premier type de contraintes est basé sur les notions de *parties obligatoires* et d'*histogramme*. Et le second type de contraintes est basé sur la notion d'*intervalles de tâches*. Nous avons présenté une modélisation, pour chacun de ces types de contraintes, en contrainte globale. Pour cela, nous avons développé des mécanismes pour maintenir les structures des données, des algorithmes permettant la propagation de ces contraintes et intégré les explications dans ces algorithmes.

Dans le chapitre 10, Nous présentons les différents événements pris en compte dans notre système pour le RCPSP dynamique, et la procédure utilisée pour la résolution du RCPSP dans le cas sur-contraint.

Chapitre 10

Expérimentations

Sommaire

Introduction

Le but principal de ce chapitre est de présenter une étude expérimentale du comportement de notre système. Nous commençons par illustrer à travers un exemple les étapes suivies par notre approche au cours de la recherche. Nous présentons ensuite quelques jeux de tests de la littérature qui serons utilisés dans nos expérimentations.

Nous nous sommes intéressé, dans un premier temps, à la la résolution du RCPSP dans le cadre statique : notre approche peut être aussi utilisée pour la résolution du RCPSP statique. Nous comparons donc notre approche aux approches existantes (section 10.3). Puis, à la résolution du RCPSP dans le cadre dynamique : après avoir présenté les différents événements pris en compte dans notre système et la procédure utilisée pour la résolution du RCPSP dans le cas sur-contraint, nous comparons notre approche avec l'approche classique qui consiste à produire, après chaque perturbation, un nouvel ordonnancement en recommençant le calcul depuis le début (section 10.4). Les critères de comparaison que nous utilisons sont : le temps de calcul et la stabilité des solutions successives.

10.1 Un exemple

Pour illustrer notre approche, considérons le RCPSP de l'exemple 37 tiré des jeux de données proposés par *Kolisch et al.* [1995].

> **Exemple 37 (Exemple illustratif) :**
> Ce problème comporte douze activités et quatre ressources. La figure 10.1 représente le graphe de précédence entre ses activités, et le tableau 10.1 présente les données de ce problème.

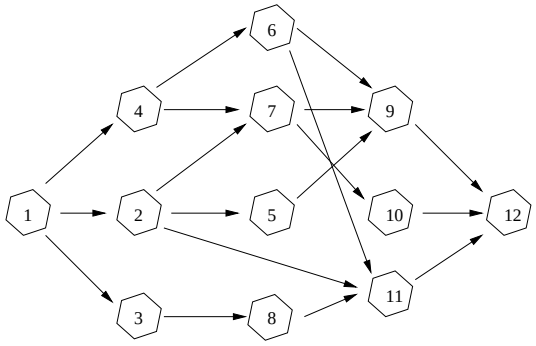


FIG. 10.1 – *Graphe de précédence de l'exemple 37*

Activité	p_i	a_{i1}	a_{i2}	a_{i3}	a_{i4}
1	0	0	0	0	0
2	7	1	0	0	0
3	4	9	0	5	6
4	3	9	0	3	0
5	1	1	9	7	0
6	2	0	0	1	8
7	5	0	9	0	0
8	5	0	5	0	10
9	7	0	0	0	5
10	8	1	0	0	8
11	8	9	8	2	0
12	0	0	0	0	0
Ressources	$R_1 = 14$	$R_2 = 16$	$R_3 = 8$	$R_4 = 14$	

TAB. 10.1 – *Données de l'exemple 37*

Nous présentons dans la figure 10.2 les transformations successives de la portion de l'arbre de recherche explorée durant la recherche d'une solution exacte de l'exemple 37.

• La portion (a) représente la première branche explorée au cours de la recherche. Partant de la racine, nous ajoutons en chaque nœud une contrainte qui correspond au premier fils f_1, f_2 ou f_3 contenant une solution réalisable. On notera en effet que, comme expliqué plus haut, certains fils ne peuvent être créés étant donné la valeur des bornes des domaines des variables. C'est le cas par exemple pour le troisième nœud de cette branche pour lequel le fils créé est f_2 ($4 \rightarrow 3$), f_1 n'étant pas un nœud réalisable. En suivant cette branche, nous obtenons une contradiction. Nous supprimons alors la contrainte la plus récente dans l'explication de cette contradiction du système, ici $5 \rightarrow 7$, et nous ajoutons sa négation.

• Nous obtenons alors une nouvelle branche de l'arbre de recherche qui définit une nouvelle portion (la portion (b)). Puisque nous savons que 5 n'est pas avant 7, nous avons deux possibilités : soit 7 est avant 5, soit les deux activités sont en parallèle. Dans cette portion, nous conservons toutes les autres contraintes et nous choisissons d'ajouter tout d'abord la contrainte $7 \rightarrow 5$. Sur la figure nous pouvons voir la partie de l'arbre, constituée des nœuds *(5,6)* et *(5,8)*, qui peut être réutilisée sans être modifiée. Notons bien que la portion de l'arbre avant le nœud *(5, 7)* est elle aussi réutilisée sans modifications. Là encore une contradiction apparaît, ce qui nous conduit à retirer la contrainte $5 \rightarrow 8$ et à ajouter sa négation.

• Dans la dernière portion, la contrainte $5 \rightarrow 8$ est remplacée par la contrainte $5 \parallel 8$ (puisque le domaine de $d_{58} = [-1, 0]$, nous ne pouvons pas avoir $8 \rightarrow 5$ et c'est donc la seule possibilité qu'il nous reste). Nous pouvons voir, sur la figure, la partie de l'arbre constituée des nœuds *(5,6)* et *(5,7)* qui peut être réutilisée sans modification. Après propagation, nous n'obtenons pas de contradiction. Nous poursuivons alors notre recherche jusqu'à obtenir une solution.

La figure 10.3 représente la solution obtenue par notre système.

10.2 Jeux de tests

Dans la littérature, plusieurs jeux de tests pour le RCPSP ont été proposés. La série des jeux de données PATT (Patterson's project scheduling problems) a été collectée par *Patterson* [1984]. Elle est constituée de 110 problèmes, comportant entre 7 et 51 activités (en moyenne 22 activités), et entre 1 et 3 ressources. Ces problèmes ont la réputation d'être faciles. En

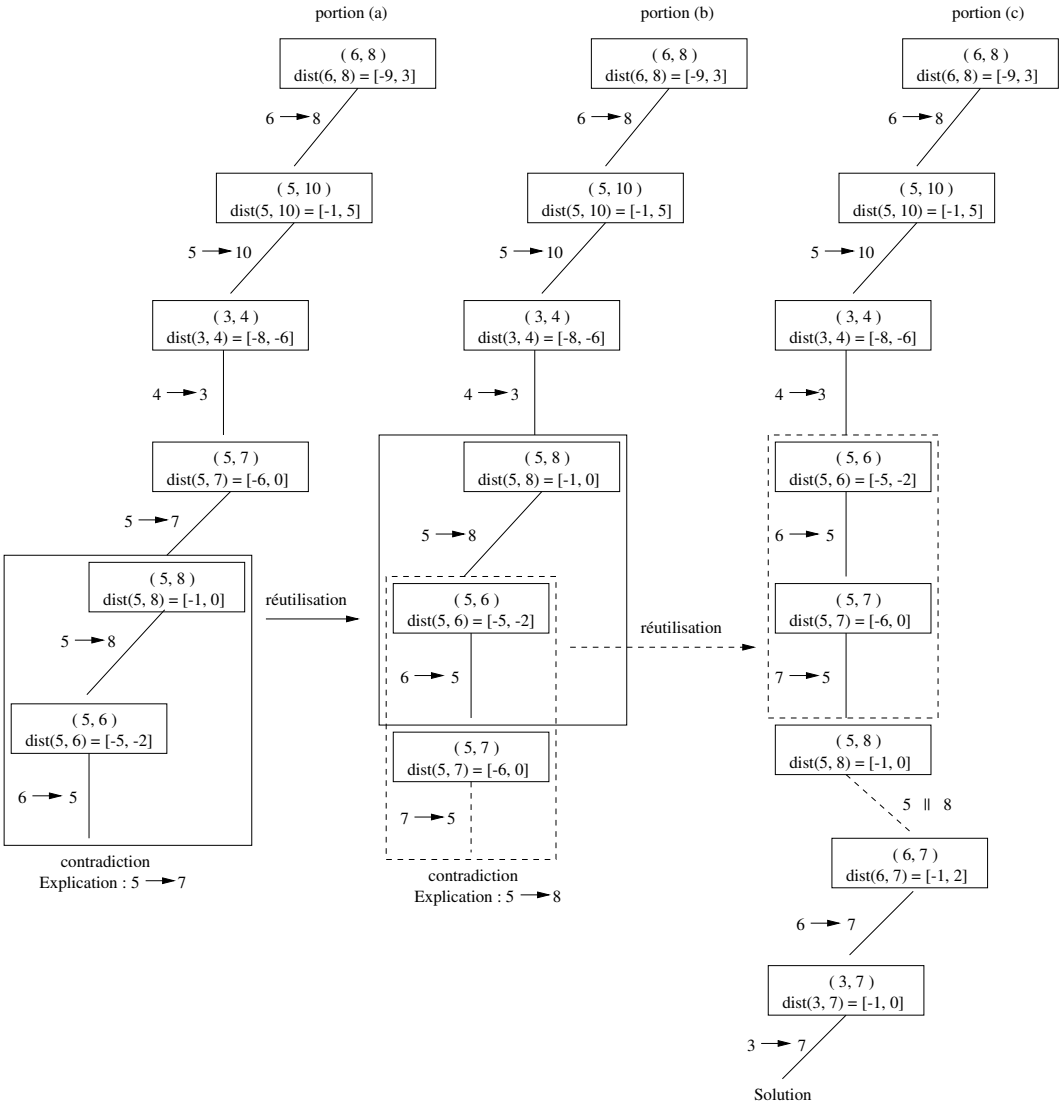


FIG. 10.2 – *Transformations de l'arbre de recherche*

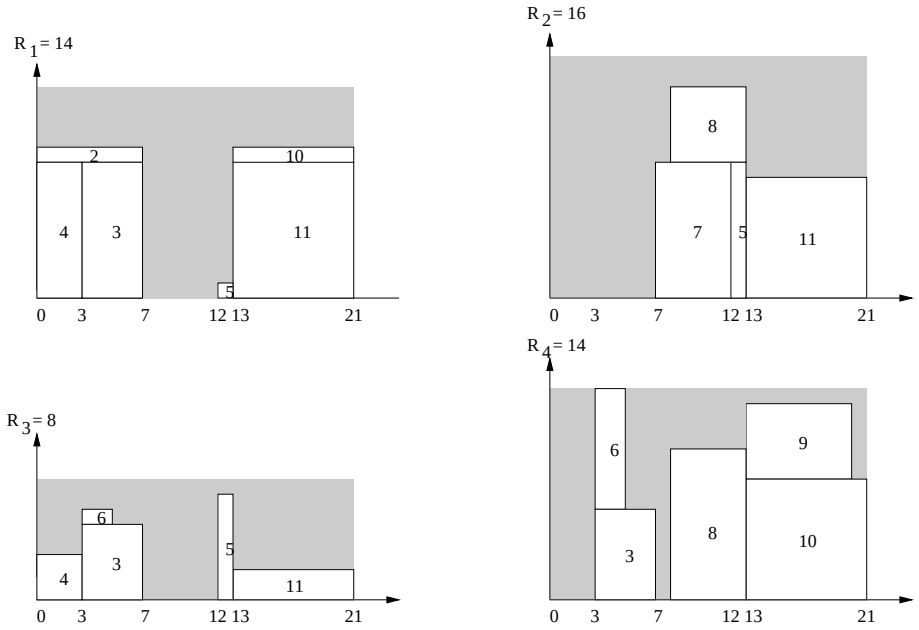


FIG. 10.3 – *La solution obtenue pour l'exemple 37*

effet, plusieurs auteurs ont réussit à les résoudre facilement : *Demeulemeester* [1992] a résolu tous ces problèmes en 0.2 secondes en moyenne, et *Caseau et Laburthe* [1994] les ont tous résolu en 3.5 secondes en moyenne.

Kolisch et al. [1995] ont proposé un générateur paramétrable de jeux de tests *ProGen*. Les jeux de tests fournis par ce générateur sont caractérisés principalement par les trois paramètres suivants :

- *NC* (*Network Complexity*) qui tient compte du nombre de précédences. Il représente le rapport entre le nombre de relations de précédence (non redondantes) dans le graphe de précédence et le nombre d'activités du problème.

- *RF* (*Resource Factor*) qui tient compte du nombre moyen de ressources utilisées par les activités. Il représente la densité du tableau $[a_{ik}]$ (a_{ik} est la quantité de ressource nécessaire pour réaliser l'activité i sur la ressource k). Ainsi, $RF = 1$ implique que chaque activité utilise toutes les ressources, et $RF = 0$ implique qu'aucune activité n'utilise de ressource.

- *RS* (*Resource Strength*) qui tient compte de la quantité moyenne de ressources utilisées par chacune des activités en fonction de la quantité de ressources disponible. Ce paramètre mesure le degré de dureté des contraintes de ressource. Ainsi, $RS = 1$ implique que l'ordonnancement de chaque activité à sa date de début au plus tôt, en ignorant les contraintes de ressource, est une solution du problème satisfaisant aussi les contraintes de ressource, et $RS = 0$ implique que pour chaque ressource k, $R_k = max_{i \in A}\{a_{ik}\}$. Les contraintes de ressources sont donc dures à satisfaire.

Parmi les jeux de données qui sont générés par le générateur *ProGen*, nous avons SMFF (*Single-Mode Full-Factorial*) et SMCP (*Single-Mode Project Scheduling*). La première série

SMFF est constituée de 480 problèmes répartis sur 48 séries de 10 problèmes. Chacun de ces 480 problèmes comporte 32 activités et 4 resources. La taille de ces 48 séries de problèmes varie entre 12 et 42 activités, et entre 1 et 6 ressources. Ces problèmes ont la réputation d'être difficiles. À notre connaissance, seule l'approche proposée par *Demeulemeester et Herroelen* [1997] permet de résoudre l'ensemble des problèmes de la série SMFF. La seconde série SMCP est constituée de 200 problèmes, de même taille (32 activités et 4 ressources), répartis sur 20 séries de 10 problèmes.

Dans le but de valider notre approche, nous avons fait des tests sur les problèmes PATT, SMFF, et SMCP. Le tableau 10.2 résume les caractéristiques de ces jeux de données.

	# problèmes	**# activités par problème**	**# ressources par problème**
PATT	110	7-51	1-3
SMCP	200	12-42	1-6
SMFF	480	32	4

TAB. 10.2 – *Caractéristiques des jeux de tests*

Dans un premier temps, nous avons mené deux séries d'expérimentations sur les deux séries de problèmes PATT et SMFF dans le cadre statique. Nous avons choisi ces deux séries car elles sont très utilisées dans la littérature. De plus, les problèmes PATT ont la réputation d'être faciles, tandis que les problèmes SMFF ont la réputation d'être difficiles.

Dans un second temps et dans le cadre dynamique, nous avons mené des expérimentations sur des problèmes de la série SMCP. Nous avons choisi cette série car les problèmes la constituant sont de différentes tailles.

10.3 Étude sur des problèmes statiques

L'objectif de cette section est d'évaluer la qualité de notre approche dans le cadre statique. Dans ce cas, notre approche peut être considérée comme une version de l'algorithme *mac-dbt* [Jussien *et al.*, 2000]. Nous avons procédé à des tests de satisfaisabilité sur les problèmes PATT et SMFF. Nous avons limité le temps d'exécution à **1800 secondes**, et pour chacun des problèmes résolus, nous avons calculé :

- le **temps d'exécution** CPU en secondes;
- le **nombre d'extensions** qui représente le nombre de contraintes de décision ajoutées au cours de la recherche jusqu'à l'obtention de la solution;
- et le **nombre de réparations** qui représente le nombre de contraintes de décision relaxées au cours de la recherche.

10.3.1 Résultats sur les problèmes PATT

Bien que ces problèmes soient faciles, notre approche a pu résoudre 48% des problèmes dans le temps imparti.

Les deux figures (10.4 et 10.5) représentent respectivement le nombre et le nombre cumulé de problèmes résolus par notre approche en fonction du temps. Sur la figure 10.5 nous pouvons

constater que notre approche a pu résoudre 12 problèmes en moins d'une seconde, et 35 en moins de 100 secondes.

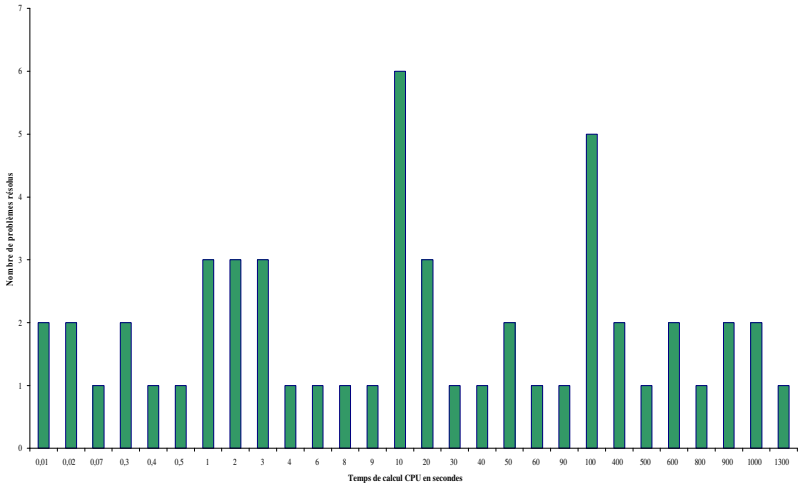


FIG. 10.4 – *Nombre de problèmes* PATT *résolus en fonction du temps* CPU

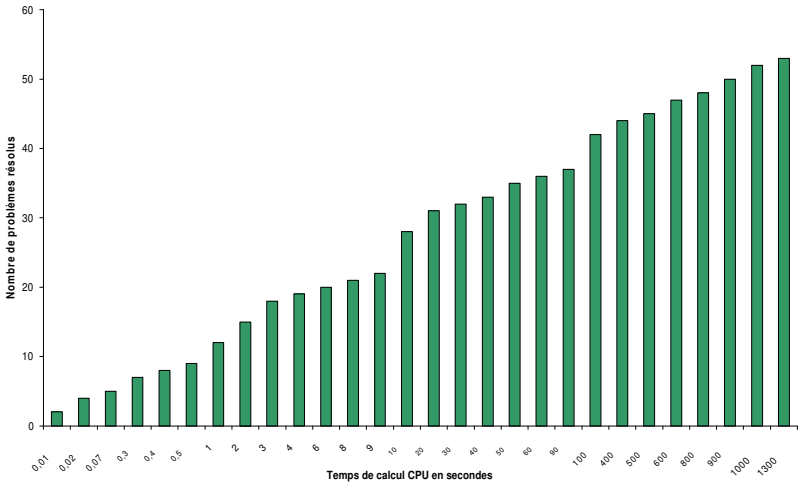


FIG. 10.5 – *Nombre cumulé de problèmes* PATT *résolus en fonction du temps* CPU

Les deux figures (10.6 et 10.7) représentent respectivement le nombre et le nombre cumulé de problèmes résolus en fonction du nombre d'extensions, *i.e.* du nombre de contraintes de décisions ajoutées au cours de la recherche . Nous constatons sur la figure 10.7 que nous avons résolu 40 problèmes en moins de 90 extensions.

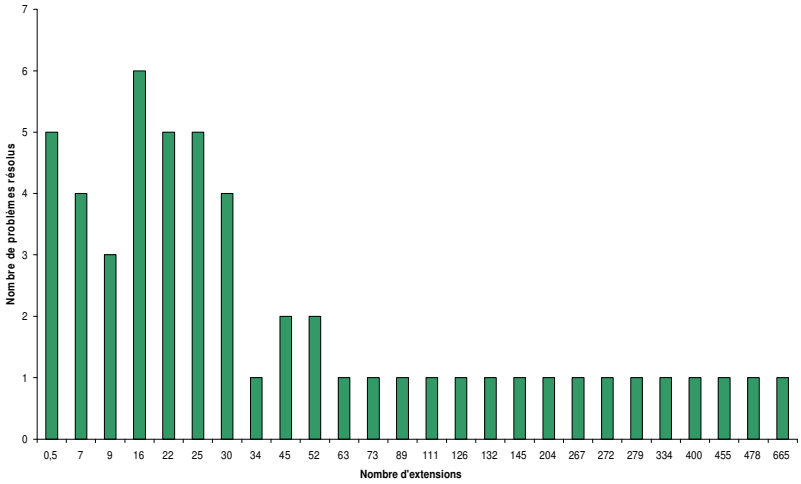


FIG. 10.6 – *Nombre de problèmes* PATT *résolus en fonction du nombre d'extensions*

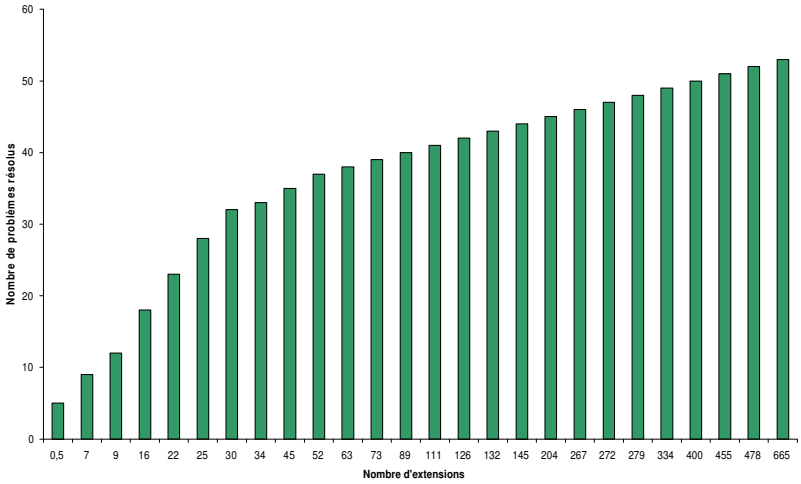


FIG. 10.7 – *Nombre cumulé de problèmes* PATT *résolus en fonction du nombre d'extensions*

Les deux figures (10.8 et 10.9) représentent respectivement le nombre et le nombre cumulé de problèmes résolus par notre approche en fonction du nombre de réparations, *i.e* du nombre de décisions remises en cause (backtracks dans le cas classique) . Dans ces deux figures nous pouvons constater que notre approche a pu résoudre 11 problèmes sans réparation, 11 autres en deux réparations, et 40 problèmes en moins de 55 réparations.

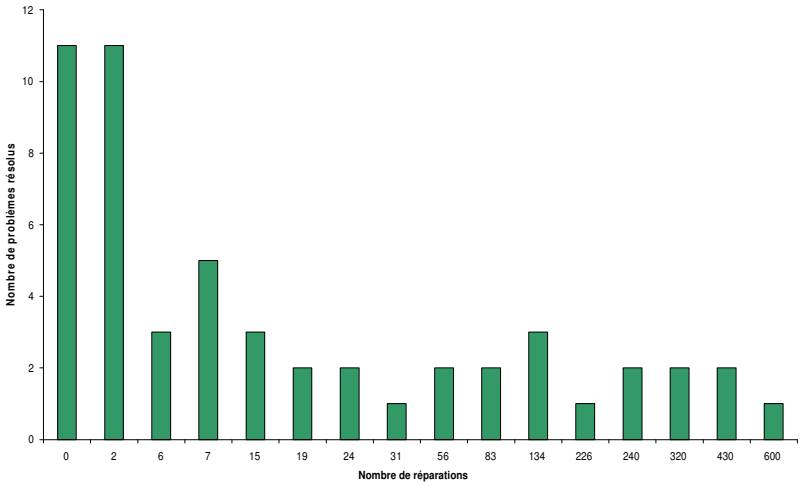


FIG. 10.8 – *Nombre de problèmes* PATT *résolus en fonction du nombre de réparations*

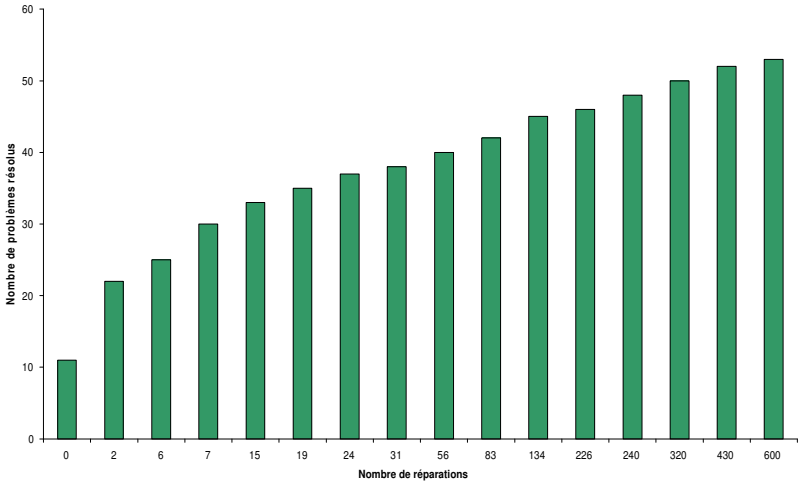


FIG. 10.9 – *Nombre cumulé de problèmes* PATT *résolus en fonction du nombre de réparations*

10.3.2 Résultats sur les problèmes SMFF

Pour la série de jeux de tests SMFF, notre approche a pu résoudre 67.3% des problèmes dans le temps imparti.

Les deux figures (10.10 et 10.11) représentent respectivement le nombre et le nombre cumulé de problèmes résolus par notre approche en fonction du temps. Nous pouvons voir sur la figure 10.11 que nous avons pu résoudre 253 problèmes en moins de 100 secondes.

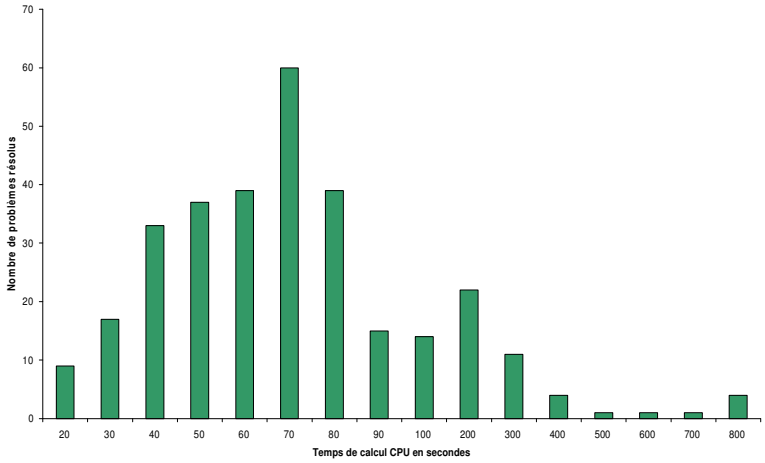


FIG. 10.10 – *Nombre de problèmes* SMFF *résolus en fonction du temps* CPU

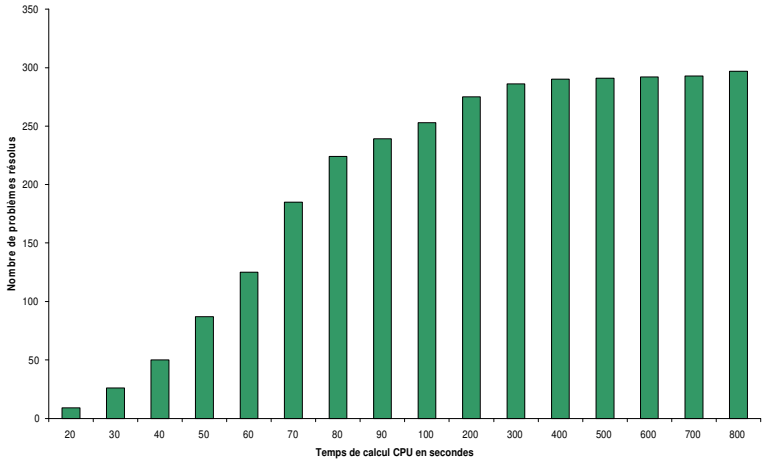


FIG. 10.11 – *Nombre cumulé de problèmes* SMFF *résolus en fonction du temps* CPU

Les deux figures (10.12 et 10.13) représentent respectivement le nombre et le nombre cumulé de problèmes résolus par notre approche en fonction du nombre d'extensions. Nous pouvons constater sur la figure 10.13 que notre approche a pu résoudre 230 problèmes en moins de 30 extensions.

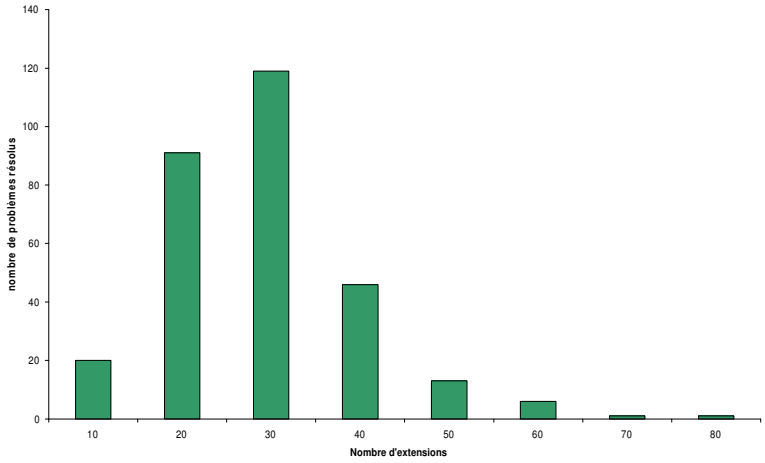


FIG. 10.12 – *Nombre de problèmes* SMFF *résolus en fonction du nombre d'extensions*

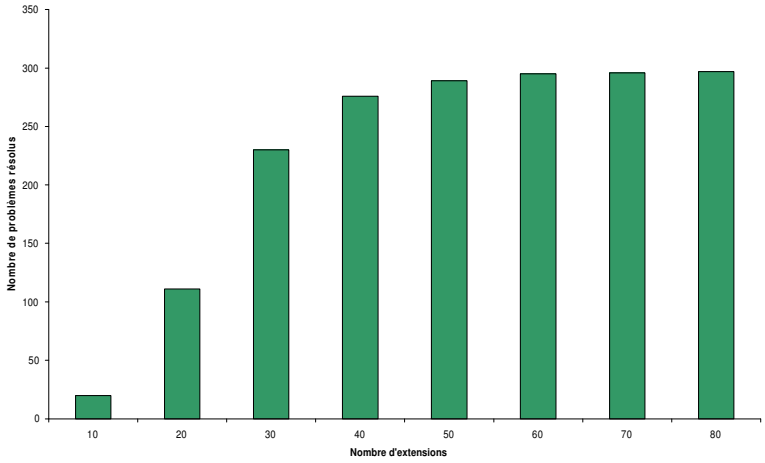


FIG. 10.13 – *Nombre cumulé de problèmes* SMFF *résolus en fonction du nombre d'extensions*

Les deux figures (10.14 et 10.15) représentent respectivement le nombre et le nombre cumulé de problèmes résolus en fonction du nombre de réparations. Sur ces deux figures nous pouvons constater que nous avons résolu 137 problèmes sans avoir recours à aucune réparation, 61 problèmes en une seule réparation, et 250 problèmes en moins de 5 réparations.

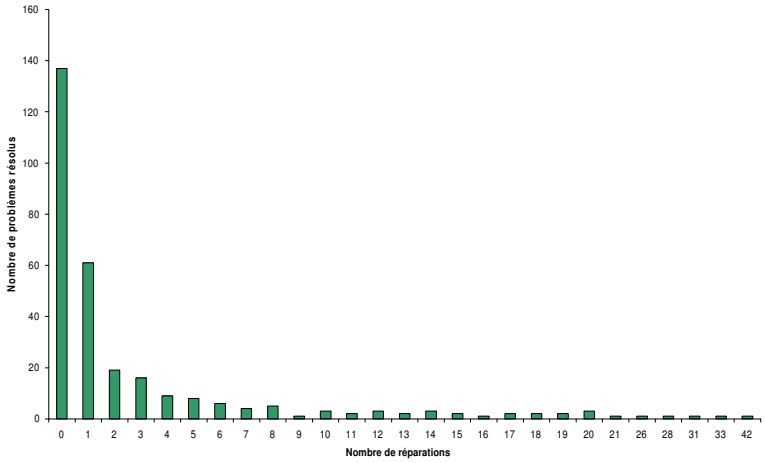


FIG. 10.14 – *Nombre de problèmes* SMFF *résolus en fonction du nombre de réparations*

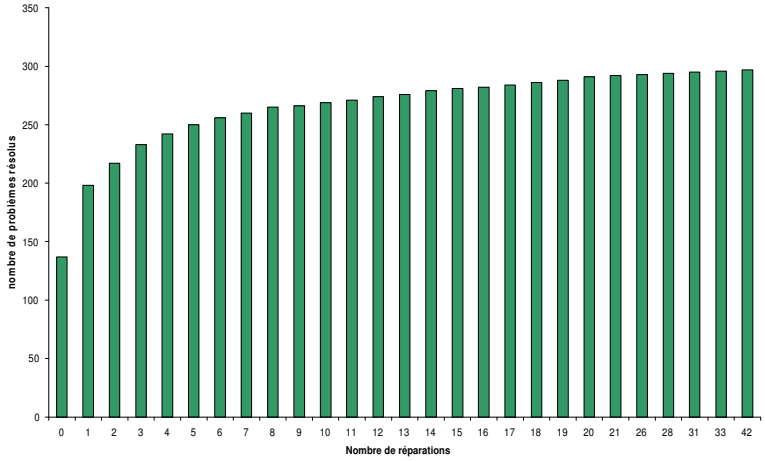


FIG. 10.15 – *Nombre cumulé de problèmes* SMFF *résolus en fonction du nombre de réparations*

Les expérimentations menées par *Caseau et Laburthe* [1994] sur les 12 premiéres séries de SMFF, ont montré que leurs approche a pu résoudre 80 problèmes de ces 120 problèmes en moins d'une seconde, et 91 en moins d'une minute. Pour notre approche, nous n'avons résolu que 86 problèmes sur les 120 premiers problèmes, 76 sont résolus en moins d'une minute.

Les deux tableaux (10.3 et 10.4) fournissent les résultats obtenus pour chaque groupe de 10 problèmes de la série SMFF. Nous présentons le temps de calcul CPU en secondes, le nombre d'extensions, et le nombre de réparations. Les symboles *moy.*, *med.*, *min.* et *max.* représentent respectivement la valeur moyenne, la médiane, la valeur minimale et la valeur maximale obtenues sur les problèmes résolus pour chacune des séries.

TAB. 10.3 – *Instances de* SMFF *(1)*

Gp	NC	RF	RS	#S	CPU(sec)		#Extensions		#Réparations	
1	1.50	0.25	0.20	10	moy. 113.8	min. 11.0	moy. 38.0	min. 15	moy. 6.8	min. 0
					med. 77.5	max. 427.0	med. 35.5	max. 74	med. 5.5	max. 28
2	1.50	0.25	0.50	10	moy. 22.5	min. 11.0	moy. 32.3	min. 29	moy. 0.4	min. 0
					med. 21.2	max. 46.0	med. 31.5	max. 38	med. 0.0	max. 2
3	1.50	0.25	0.70	10	moy. 22.2	min. 13.8	moy. 37.1	min. 30	moy. 0.3	min. 0
					med. 24.0	max. 34.6	med. 37.0	max. 45	med. 0.0	max. 0
4	1.50	0.25	1.00	10	moy. 23.7	min. 6.2	moy. 33.8	min. 20	moy. 0.6	min. 0
					med. 18.6	max. 56.0	med. 33.0	max. 54	med. 0.0	max. 5
5	1.50	0.50	0.20	3	moy. 454.7	min. 398.7⋆	moy. 85.0	min. 47	moy. 46.0	min. 17
					med. 454.7	max. 510.7	med. 69.0	max. 139	med. 22.0	max. 100
6	1.50	0.50	0.50	5	moy. 257.2	min. 79.0	moy. 85.6	min. 61	moy. 43.6	min. 15
					med. 263.8	max. 409.2	med. 78.0	max. 136	med. 39.0	max. 95
7	1.50	0.50	0.70	10	moy. 716.6	min. 14.6	moy. 49.6	min. 35	moy. 11.4	min. 1
					med. 39.9	max. 360.5	med. 42.0	max. 131	med. 5.0	max. 77
8	1.50	0.50	1.00	10	moy. 26.3	min. 15.2	moy. 36.5	min. 28	moy. 1.6	min. 0
					med. 22.5	max. 53.9	med. 36.5	max. 56	med. 0.5	max. 6
9	1.50	0.75	0.20	1	moy. 1704.4	min. 1704.4	moy. 88.0	min. 88	moy. 37.0	min. 37
					med. 1704.4	max. 1704.4	med. 88.0	max. 88	med. 37.0	max. 37
10	1.50	0.75	0.50	2	moy. 33.2	min. 15.5	moy. 43.5	min. 30	moy. 7.0	min. 0
					med. 33.2	max. 51.0	med. 43.5	max. 57	med. 7.0	max. 14
11	1.50	0.75	0.70	5	moy. 215.6	min. 26.8	moy. 69.4	min. 33	moy. 22.2	min. 1
					med. 56.7	max. 679.1	med. 47.0	max. 143	med. 8.0	max. 110
12	1.50	0.75	1.00	10	moy. 354.2	min. 6.7	moy. 39.4	min. 25	moy. 4.8	min. 0
					med. 14.7	max. 182.6	med. 32.5	max. 86	med. 0.5	max. 44
14	1.50	1.00	0.50	2	moy. 170.1	min. 56.4	moy. 72.0	min. 43	moy. 39.0	min. 11
					med. 170.1	max. 283.8	med. 72.0	max. 110	med. 39.0	max. 67
15	1.50	1.00	0.70	8	moy. 137.8	min. 13.8	moy. 54.5	min. 37	moy. 14.4	min. 0
					med. 61.7	max. 700.6	med. 47.0	max. 116	med. 8.5	max. 64
16	1.50	1.00	1.00	10	moy. 17.9	min. 9.5	moy. 35.0	min. 19	moy. 0.9	min. 0
					med. 19.1	max. 24.8	med. 39.5	max. 43	med. 0.0	max. 3
17	1.80	0.25	0.20	10	moy. 80.6	min. 11.1	moy. 27.7	min. 15	moy. 1.3	min. 0
					med. 24.5	max. 382.4	med. 29.0	max. 43	med. 0.5	max. 7
18	1.80	0.25	0.50	10	moy. 41.9	min. 12.2	moy. 26.1	min. 17	moy. 0.5	min. 0
					med. 36.6	max. 100.3	med. 23.0	max. 46	med. 0.0	max. 2
19	1.80	0.25	0.70	10	moy. 33.2	min. 6.0	moy. 31.6	min. 21	moy. 0.9	min. 0
					med. 31.8	max. 61.5	med. 29.5	max. 53	med. 1.9	max. 4
20	1.80	0.25	1.00	10	moy. 20.3	min. 9.5	moy. 27.2	min. 18	moy. 0.2	min. 0
					med. 17.5	max. 42.5	med. 27.5	max. 38	med. 0.0	max. 1
21	1.80	0.50	0.20	2	moy. 115.9	min. 115.9 ⋆	moy. 68.0	min. 23	moy. 34.0	min. 1
					med. 115.9	max. 115.9	med. 68.0	max. 113	med. 34.0	max. 67
22	1.80	0.50	0.50	8	moy. 250.3	min. 11.1	moy. 61.5	min. 27	moy. 25.1	min. 3
					med. 72.5	max. 957.1	med. 45.5	max. 138	med. 6.5	max. 104

⋆ indique que les valeurs calculées pour le *CPU* tiennent compte de tous les problèmes résolus de cette série sauf pour un problème.

TAB. 10.4 – *Instances de* SMFF *(2)*

Gp	NC	RF	RS	#S	CPU(sec)		#Extensions		#Réparations	
23	1.80	0.50	0.70	10	moy. 45.9	min. 15.8	moy. 34.9	min. 22	moy. 2.2	min. 0
					med. 49.2	max. 90.2	med. 37.5	max. 44	med. 1.0	max. 7
24	1.80	0.50	1.00	10	moy. 19.6	min. 5.0	moy. 27.6	min. 13	moy. 0.3	min. 0
					med. 18.0	max. 38.3	med. 27.5	max. 39	med. 0.0	max. 2
26	1.80	0.75	0.50	5	moy. 115.0	min. 19.0	moy. 56.4	min. 31	moy. 23.0	min. 0
					med. 90.5	max. 219.8	med. 47.0	max. 103	med. 15.0	max. 67
27	1.80	0.75	0.70	9	moy. 61.6	min. 16.6	moy. 40.3	min. 25	moy. 8.0	min. 0
					med. 34.3	max. 305.5	med. 39.0	max. 69	med. 2.0	max. 30
28	1.80	0.75	1.00	10	moy. 30.4	min. 13.1	moy. 36.6	min. 26	moy. 1.9	min. 0
					med. 22.9	max. 86.3	med. 39.5	max. 46	med. 1.0	max. 5
31	1.80	1.00	0.70	7	moy. 68.7	min. 17.4	moy. 42.0	min. 32	moy. 5.3	min. 0
					med. 25.7	max. 307.1	med. 38.0	max. 55	med. 3.0	max. 24
32	1.80	1.00	1.00	10	moy. 56.4	min. 7.5	moy. 36.1	min. 19	moy. 5.0	min. 0
					med. 23.3	max. 358.5	med. 33.0	max. 88	med. 1.0	max. 44
33	2.10	0.25	0.20	10	moy. 98.5	min. 33.1	moy. 27.7	min. 21	moy. 1.3	min. 0
					med. 100.1	max. 177.0	med. 30.0	max. 35	med. 1.0	max. 4
34	2.10	0.25	0.50	10	moy. 43.2	min. 7.8	moy. 25.2	min. 18	moy. 0.2	min. 0
					med. 34.8	max. 99.2	med. 25.0	max. 35	med. 0.0	max. 2
35	2.10	0.25	0.70	9	moy. 29.7	min. 9.1	moy. 26.1	min. 17	moy. 0.5	min. 0
					med. 30.5	max. 55.7	med. 25.0	max. 39	med. 0.0	max. 2
36	2.10	0.25	1.00	10	moy. 29.7	min. 13.5	moy. 28.5	min. 19	moy. 0.6	min. 0
					med. 28.9	max. 58.3	med. 28.0	max. 37	med. 1.0	max. 1
37	2.10	0.50	0.20	3	moy. 1044.3	min. 132.2	moy. 52.0	min. 26	moy. 14.6	min. 0
					med. 1344.8	max. 1655.8	med. 61.0	max. 69	med. 19.0	max. 25
38	2.10	0.50	0.50	8	moy. 167.2	min. 23.6	moy. 43.2	min. 29	moy. 14.5	min. 1
					med. 127.1	max. 415.6	med. 39.5	max. 58	med. 13.5	max. 32
39	2.10	0.50	0.70	10	moy. 96.0	min. 14.2	moy. 33.6	min. 16	moy. 3.2	min. 0
					med. 42.1	max. 458.3	med. 30.5	max. 57	med. 1.0	max. 22
40	2.10	0.50	1.00	10	moy. 31.8	min. 19.5	moy. 30.1	min. 19	moy. 2.3	min. 0
					med. 25.9	max. 68.6	med. 30.5	max. 42	med. 0.5	max. 12
42	2.10	0.75	0.50	4	moy. 96.0	min. 25.7	moy. 39.7	min. 31	moy. 9.7	min. 1
					med. 76.3	max. 205.9	med. 33.5	max. 61	med. 6.0	max. 26
43	2.10	0.75	0.70	5	moy. 124.8	min. 29.0	moy. 39.2	min. 22	moy. 15.8	min. 4
					med. 82.2	max. 335.3	med. 31.0	max. 73	med. 7.0	max. 53
44	2.10	0.75	1.00	10	moy. 28.7	min. 8.0	moy. 27.1	min. 15	moy. 0.6	min. 0
					med. 26.9	max. 58.1	med. 28.0	max. 40	med. 0.0	max. 3
47	2.10	1.00	0.70	7	moy. 98.8	min. 58.9	moy. 38.8	min. 21	moy. 11.2	min. 5
					med. 80.6	max. 156.1	med. 38.0	max. 51	med. 9.0	max. 28
48	2.10	1.00	1.00	10	moy. 62.9	min. 7.9	moy. 36.9	min. 16	moy. 9.2	min. 0
					med. 26.4	max. 392.3	med. 29.5	max. 111	med. 0.0	max. 90

Notre approche ne résout aucun problème des séries : 13, 25, 29, 30, 41, 45 et 46. Et nous n'avons pu résoudre qu'un seul problème de la série 9, et ce, en plus de 1700 secondes.

D'après les résultats des tableaux (10.3 et 10.4), nous pouvons déduire le pourcentage des problèmes résolus suivant les valeurs des paramètres NC, RF, et RS (*cf.* tableau 10.5). Nous pouvons constater que :

- le pourcentage des problèmes résolus est presque le même pour toutes les valeurs du paramètre NC. Cela signifie que le nombre de relations de précédence n'a aucune influence sur notre approche;
- pour le paramètre RF, le pourcentage des problèmes résolus passe de 99.1%, lorsque chaque activité utilise en moyen une seule ressource ($RF = 0.25$), à 45%, lorsque chaque activité utilise les quatres ressources ($RF = 1.00$);
- concernant le paramètre RS, le pourcentage des problèmes résolus passe de 32.5%, lorsque les activités n'utilisent en moyenne que 20% des capacités des ressources ($RS = 0.20$), à 100%, lorsque les activités utilisent en moyenne toutes les capacités des ressources ($RS = 1.00$).

Ces deux derniers résultats peuvent être expliqués par le fait que lorsque nous augmentons la valeur du paramétre RF et/ou nous diminuons la valeur du paramétre RS, nous avons de plus en plus de conflits de ressources à résoudre. En effet, nous pouvons constater, dans les tableaux (10.3 et 10.4), que le temps de résolution des problèmes pour lesquels la valeur $RF = 0.25$ et $RS = 1.00$ est compris entre 6.2 et 58.3 secondes.

NC			**RF**				**RS**			
1.50	1.80	2.10	0.25	0.50	0.75	1.00	0.20	0.50	0.70	1.00
66.2%	69.3%	66.2%	99.1%	71.6%	50.8%	45%	32.5%	53.3%	83.3%	100%

TAB. 10.5 – *Nombre de problèmes résolu* SMFF

10.3.3 Comparaison des résultats

Nous présentons dans le tableau 10.6 une synthése des résultats expérimentaux. Par exemple, lorsque nous allouons 100 secondes par problème, nous pouvons résoudre 38% des problèmes de la série PATT et 53% des problèmes de la série SMFF.

	CPU			**Extensions**			**Réparations**		
PATT	1 s.	100 s.	500 s.	9 ex.	111 ex.	400 ex.	2 rep.	24 rep.	240 rep.
	10%	38%	41%	11%	37%	45%	20%	34%	44%
SMFF	30 s.	100 s.	500 s.	20 ex.	40 ex.	60 ex.	1 rep.	15 rep.	26 rep.
	5%	53%	60%	23%	57%	61%	41%	58%	61%

TAB. 10.6 – *Quelques résultats obtenus sur les jeux de tests* PATT *et* SMFF

Les problèmes PATT sont des problèmes fortement disjonctifs [Baptiste et Le Pape, 1997]. Une large partie de ces disjonctions proviennent des contraintes de précédence entre les activités. Ainsi, nous avons jusqu'à 90% d'activités qui peuvent être reliées par une relation de précédence. Le nombre d'activités susceptibles d'être exécutées simultanément est alors

en moyenne assez faible. Il est donc normal que notre approche utilisant des mécanismes de déduction complexes ait un effet limité sur la résolution de ces instances.

D'après les résultats obtenus, nous pouvons constater que pour notre approche la série SMFF est plus facile à résoudre que la série PATT (pour SMFF : 61% des problèmes ont été résolus en moins de 26 réparations, alors que pour PATT : 33% des problèmes ont été résolus en moins de 24 réparations). Nous constatons aussi que le nombre d'extensions (pour les deux séries) est très faible par rapport aux approches standards n'utilisant pas de réparations. Le surcoût apporté par le calcul et l'utilisation des explications ne semble pas être compensé par les améliorations de la résolution.

Nous n'avons pas cherché à développer une démarche vraiment spécifique pour le cadre statique. En effet, nous n'avons pas utilisé plusieurs techniques issues de la recherche opérationnelle (*i.e.* les règles de dominance *left-shift* utilisée par *Caseau et Laburthe* [1994], ou *Cut-Set* utilisée par *Demeulemeester et Herroelen* [1997]) pour accélérer la recherche, car l'utilisation de ces techniques peut influencer la stabilité des solutions successives dans le cas dynamique.

Bien que notre approche soit dédiée à la résolution du RCPSP dynamique, ces premiers résultats sur le cas statique sont satisfaisants.

10.4 Étude sur des problèmes dynamiques

Après avoir présenté les différents aléas pris en compte dans notre système pour le RCPSP dynamique, nous présentons les différents paramètres utilisées pour évaluer notre approche. Et enfin, les résultats des expérimentations menées dans le cadre dynamique.

10.4.1 Événements pris en compte dans notre système

Cette section liste les différents événements pouvant être pris en compte dans notre système. Ils sont de trois types : les événements temporels, les événements liés aux activités et les événements liés aux ressources.

10.4.1.1 Événements temporels

Les événements temporels sont tous les événements pouvant être modélisés par l'ajout, le retrait ou la modification de contraintes temporelles telles que celles rencontrées dans le RCPSP classique, *i.e.* des précédences, des disjonctions, des contraintes de parallélisme.

Ajout de contraintes Le tableau 10.7 nous donne la contrainte à ajouter pour chacune des relations temporelles reliant deux activités i et j.

Relation temporelle	Contrainte
i $\rightarrow$ j	$d_{ij} \geq 0$
i $\leftrightarrow$ j	$d_{ij} \times d_{ji} \leq 0$
i $\parallel$ j	$d_{ij} \times d_{ji} > 0$

TAB. 10.7 – *Traduction des événements en contraintes*

Exemple 38 (Ajout d'une précédence) :
Reprenons l'exemple 37, et supposons que nous avons un événement obligeant l'activité 6 à ne commencer qu'après la date de fin de l'activité 3. Ceci correspond à l'ajout de la contrainte $3 \rightarrow 6$.

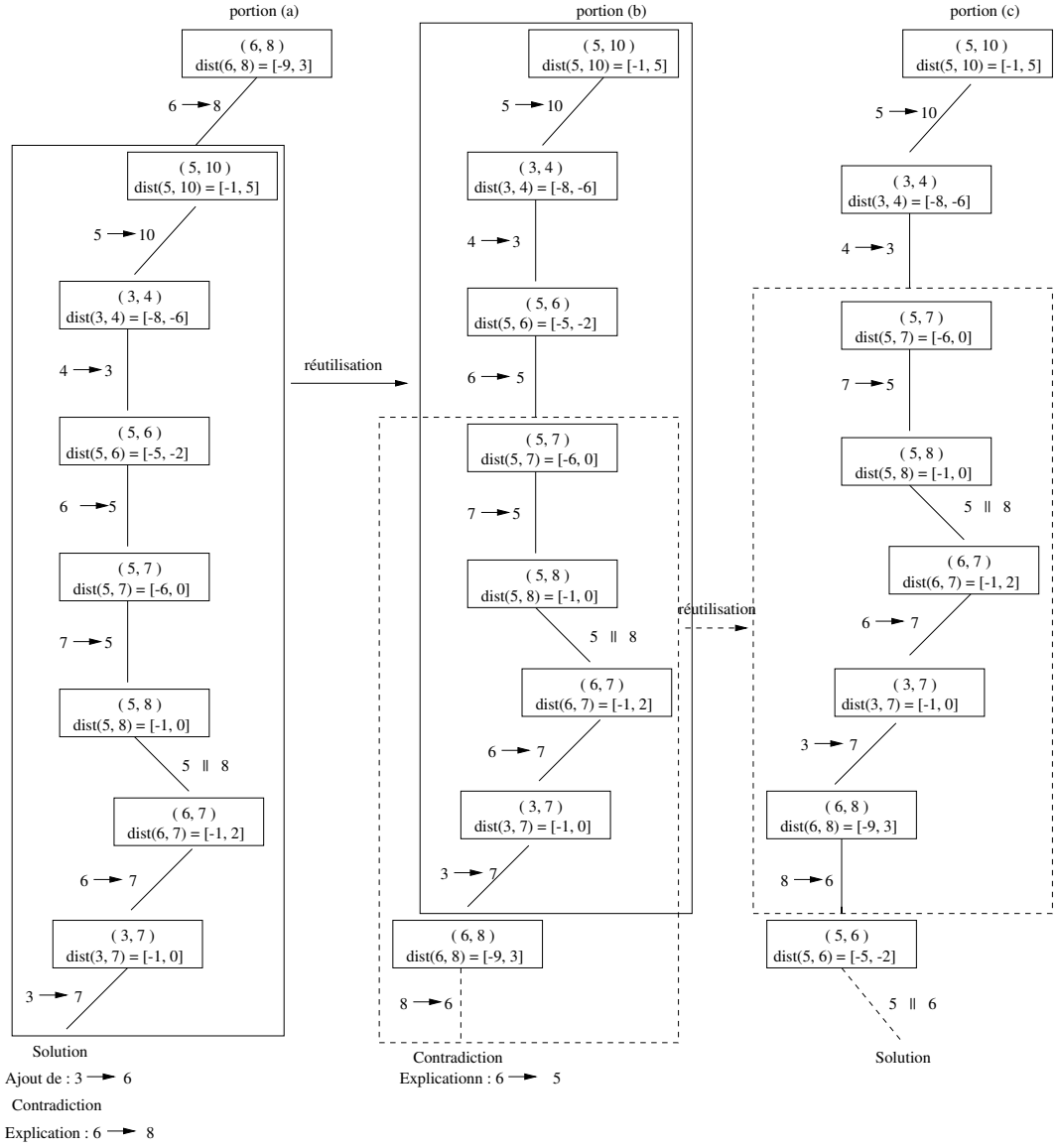


FIG. 10.16 – *Ajout d'une contrainte de précédence*

La figure 10.16 présente les transformations successives des branches explorées lors de la recherche d'une nouvelle solution. Sur cette figure, les parties de l'arbre encadrées sont les parties réutilisées sans être modifiées.

Nous pouvons constater que deux modifications suffisent dans ce cas pour obtenir une nouvelle solution : le retrait de $6 \rightarrow 8$ remplacé par l'ajout de la contrainte $8 \rightarrow 6$, puis le

retrait de $6 \rightarrow 5$ remplacé par la contrainte $5 \parallel 6$.

La nouvelle solution obtenue est représentée par la figure 10.17.

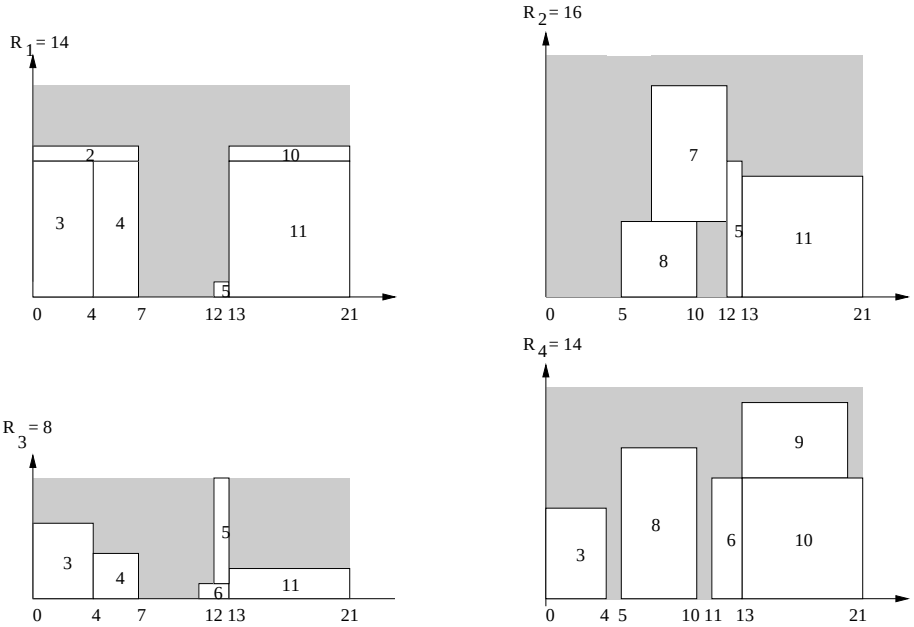


FIG. 10.17 – *Nouvelle solution obtenue pour l'exemple 37 après l'ajout de la contrainte* $3 \rightarrow 6$

Retrait de contraintes Notre système est capable de retirer dynamiquement tous les événements temporels vus précédemment. Il suffit pour cela de retirer du système les contraintes associées à ces événements.

Modification de contraintes Toutes les contraintes définies précédemment peuvent être modifiées. La procédure consiste à retirer la contrainte associée à l'ancien événement temporel et à ajouter la contrainte associée au nouvel événement temporel.

10.4.1.2 Evénements se rapportant aux activités

Notre système offre la possibilité d'ajouter, de supprimer, et de modifier une activité.

Ajout d'une nouvelle activité Une activité i est définie par sa durée opératoire p_i, ses besoins en ressources $a_{i1}, a_{i2}, .., a_{ir}$,

La procédure permettant d'ajouter une telle activité consiste à :

- créer une nouvelle variable t_i et à la connecter au réseau de contraintes;
- ajouter les contraintes temporelles associées;
- ajouter les contraintes de disjunction résultantes de la limitation des capacités de ressources;
- insérer la variable t_i dans les contraintes de ressources *parties obligatoires* et *intervalle de tâches*;

• créer les contraintes de ressources *intervalles de tâches* qui auront i comme activité de début ou de fin.

Retrait d'une activité Pour retirer une activité, il suffit de :

• déconnecter la variable t_i du réseau de contraintes;

• retirer toutes les contraintes temporelles dont une au moins des variables est t_i, d_{ij}, ou d_{ji};

• retirer la variable t_i des contraintes de ressources *parties obligatoires* et *intervalle de tâches*;

• retirer toutes les contraintes de ressources *intervalle de tâches* qui ont i comme activité de début ou de fin.

Modification d'une activité La modification de la date opératoire et/ou de la quantité de ressource d'une activité consiste à retirer cette activité et à ajouter une nouvelle activité prenant en compte les modifications.

10.4.1.3 Evénements se rapportant aux ressources

Notre système permet aussi d'ajouter, retirer ou modifier une ressource.

Ajout d'une nouvelle ressource Une ressource est définie par sa capacité et ses besoins en chaque ressource.

La procédure ajoutant une ressource consiste à :

• ajouter les contraintes disjunctives dûes à la capacité limitée de cette nouvelle ressource;

• ajouter la contrainte de ressources *parties obligatoires* et les contraintes *intervalle de tâches* liées à cette ressource.

Retrait d'une ressource Pour retirer une ressource du problème, il suffit de :

• supprimer les contraintes disjunctives résultantes de cette ressource;

• supprimer la contrainte de ressources *parties obligatoires* et les contraintes *intervalle de tâches* liées à cette ressource.

Modification d'une ressource La modification de la capacité d'une ressource consiste à retirer cette ressource et à ajouter une nouvelle ressource prenant en compte la nouvelle modification.

10.4.1.4 Le cas sur-contraint

Il est possible que l'arrivée d'un événement inattendu aboutisse à un problème sans solution. Dans ce cas, notre système est capable de déterminer une explication de cette contradiction. L'utilisateur peut ensuite choisir dans cette explication les contraintes à retirer ou à modifier jusqu'à obtenir une solution.

> **Exemple 39 (Un exemple sans solution) :**
> Reprenons l'exemple 37, et supposons maintenant que le nouvel événement est : l'activité 3 doit être placée après l'activité 10, ce qui correspond à l'ajout de la contrainte $10 \rightarrow 3$.

En ajoutant la contrainte $10 \rightarrow 3$ au système, nous obtenons une contradiction. Le nouveau problème ne possède pas de solution. Dans notre exemple, l'explication de cette contradiction est l'ensemble de contraintes suivant : $Expl = \{C_{max} \leq 21,\ 2 \rightarrow 5,\ 2 \rightarrow 7,\ 2 \rightarrow 11,\ 3 \rightarrow 8,\ 5 \rightarrow 9,\ 7 \rightarrow 9,\ 7 \rightarrow 10,\ 8 \rightarrow 11,\ 10 \rightarrow 3\}$.

L'utilisateur peut donc choisir parmi ces contraintes, celles à relaxer jusqu'à l'obtention d'une solution. Si l'utilisateur choisi d'augmenter la date de fin du projet, c'est à dire de retirer la contrainte $C_{max} \leq 21$, il obtiendra une solution optimale de $C_{max} = 37$.

La figure 10.18 représente la nouvelle solution trouvée par notre système.

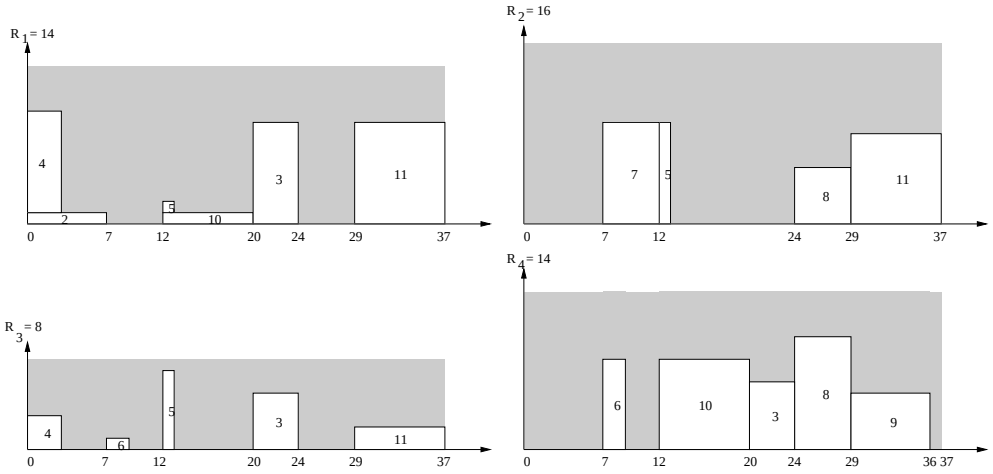


FIG. 10.18 – *Solution optimale obtenue après augmentation de* C_{max}

10.4.2 Évaluation d'un système d'ordonnancement dynamique

Il est très difficile d'évaluer un système d'ordonnancement dynamique, surtout lorsque celui-ci n'est pas dédié à une ou plusieurs applications réelles bien précises. Cependant, deux questions se posent :

• **Comment peut-on valider un système d'ordonnancement dynamique?**

La validation d'un système d'ordonnancement dynamique peut se faire en le comparant à un ou plusieurs systèmes dynamiques permettant de résoudre les mêmes types de problèmes en les soumettant à des jeux de tests ou des données réelles.

Nous avons rencontré deux difficultés pour valider notre système. La première est qu'il n'existe pas d'approche permettant la résolution optimale du RCPSP dans un cadre dynamique. Et la seconde difficulté est qu'il n'existe pas de problèmes de référence pour le RCPSP dynamique. Pour cela, nous avons décidé de comparer notre approche à l'approche classique qui consiste à recommencer le calcul à nouveau après chaque perturbation. En plus, nous avons créé nos propres jeux de tests en nous basant sur des jeux de tests existants pour le RCPSP statique, plus précisément sur la série de problèmes SMCP (*cf.* section 10.2 page 115). Aussi, ces jeux de tests ont été perturbés de manière à être proches de la réalité. Nous avons donc fait des tests qui concernent l'ajout d'une série d'activités, le retrait d'une série d'activités, l'ajout d'une série de relations de précédence et la modification d'une série de relations de précédence.

• Comment mesurer les performances d'un système d'ordonnancement dynamique ?

Les performances d'un système d'ordonnancement dynamique peuvent être mesurées en terme de temps de calcul et/ou en terme de qualité des solutions obtenues. Cette dernière peut être obtenue en comparant la nouvelle solution à la solution courante ou à la solution initiale. Plusieurs mesures ont été proposées dans la littérature (*cf.* section 3.3.2 page 33).

Nous avons évalué la performance de notre système en étudiant son évolution au cours du temps. Nous avons donc mesuré la qualité des solutions obtenues successivement après chaque perturbation.

Si nous notons par t_i^k la date de début de l'activité i dans la solution S_k, les mesures de performance que nous avons utilisées sont :

- DIFF(S_p, S_q) qui représente le nombre d'activités qui ont changé de date de dédut lorsque nous passons de la solution S_p à la solution S_q. Cette mesure peut être formulée par :

$$DIFF(S_p, S_q) = \sum_{i \in A} \delta_i(S_p, S_q) \text{ où } \delta_i(S_p, S_q) = 1 \text{ si } t_i^p \neq t_i^q \text{ et } 0 \text{ sinon}$$

- POSI(S_p, S_q) qui représente le nombre de positions relatives qui ont changé lorsque nous passons de la solution S_p à la solution S_q. Cette mesure est formulée par :

$$POSI(S_p, S_q) = \sum_{(i,j) \in A \times A,\ i<j} \gamma_j^i(S_p, S_q) \text{ où } \gamma_j^i(S_p, S_q) = 1 \text{ si la position relative}$$

$$\text{entre } i \text{ et } j \text{ a changé et } 0 \text{ sinon}$$

- SDIF(S_p, S_q) qui représente la somme des perturbations lorsque nous passons de la solution S_p à la solution S_q. Cette mesure peut être formulée par :

$$SDIF(S_p, S_q) = \sum_{i \in A} |t_i^p - t_i^q|$$

- MDIF(S_p, S_q) qui représente la perturbation maximale lorsque nous passons de la solution S_p à la solution S_q. Cette mesure peut être formulée par :

$$MDIF(S_p, S_q) = \max_{i \in A}\{|t_i^p - t_i^q|\}$$

10.4.3 Protocole expérimental

Les perturbations que nous considérons dans nos expérimentations sont : l'ajout d'une série d'activités, le retrait d'une série d'activités, l'ajout d'une série de contraintes de précédence, et le retrait d'une série de contraintes de précédence.

Dans ce qui suit, nous notons par P_0 le problème initial de la série, et P_i le problème obtenu à la *ième* perturbation.

10.4.3.1 Ajout d'une série d'activités

Les séries de problèmes sont construites de la façon suivante :

- le problème initial P_0 est construit à partir d'un problème de la série SMCP dans lequel nous retirons aléatoirement des activités, et les contraintes de précédence les reliant. Aussi, nous ajoutons à ce problème des contraintes de précédence qui relient les prédécesseurs de l'activité sélectionnée à l'activité fictive 1 et ses successeurs à l'activité fictive n (pour que le problème initial soit un RCPSP);
- le problème P_i est construit à partir du problème P_{i-1} en lui ajoutant aléatoirement une activité non encore séléctionnée parmi celles qui ont été retirées. Nous ajoutons aussi les relations de précédence, parmi celles qui ont été retirées, qui relient l'activité séléctionnée aux activités du problème P_{i-1}.

10.4.3.2 Retrait d'une série d'activité

Chaque série de problèmes est construite comme suit :

- le problème initial P_0 est un problème de la série SMCP;
- le problème P_i est obtenu à partir du problème P_{i-1} en retirant aléatoirement une activité et les relations de précédence qui relient cette activité aux autres activités, et en ajoutant des contraintes de précédence reliant les successeurs et les prédécesseurs de l'activité retirée aux activités fictives.

10.4.3.3 Ajout d'une série de contraintes de précédence

Chaque série est construite de la façon suivante :

- le problème initial P_0 est construit à partir d'un problème de la série SMCP dans lequel nous retirons aléatoirement des contraintes de précédence. Nous ajoutons à ce problème des contraintes de précédence qui relient les activités reliées par les précédences sélectionnées aux activités fictives (pour que le problème initial soit un RCPSP);
- le problème P_i est construit à partir du problème P_{i-1} en lui ajoutant la *ième* précédence retirée.

Notons que, dans certains cas, la solution courante reste valide pour la contrainte de précédence sélectionnée parmis les contraintes de la série considérée. Dans nos expérimentations nous n'avons gardé que les séries pour lesquelles les solutions courantes produites à chaque étape (par les deux approches) ne vérifient pas la contrainte de précédence sélectionnée.

10.4.3.4 Retrait d'une série de contraintes de précédence

Les séries de problèmes sont construites comme suit :

- le problème initial P_0 est un problème de la série SMCP;
- le problème P_i est construit à partir du problème P_{i-1} en lui retirant aléatoirement une précédence et en ajoutant sa négation.

Nous avons fait des tests sans ajout de la négation de la contrainte retirée, il s'avére que la date de fin du projet (C_{max}) ne diminue qu'après le retrait de plusieurs précédences. Ce qui n'est pas profitable car si le C_{max} reste le même, nous pouvons garder la solution courante. De plus, il est difficile de trouver des séquences pour lesquelles le C_{max} diminue après chaque retrait. Nous avons donc décidé, après chaque retrait d'une précédence, d'ajouter sa négation. Dans ce cas nous sommes sûrs que la solution courante ne sera plus valide.

10.4.4 Résultats expérimentaux

Les expérimentations que nous avons menées sont faites sur des problèmes de la série SMCP. Nous avons classé ces problèmes selon leurs tailles : des problèmes comportant 12 activités, 22 activités, puis 32 activités.

Nous avons construit, pour chaque type de perturbation, un grand nombre des séries de problèmes. Ainsi pour :

- l'ajout d'une série d'activités, nous avons construit 60 séries de problèmes. 10 séries de problèmes pour chacune des classes suivantes : 12 activités et 4 ressources, 22 activités et 4 ressources, 32 activités et une ressource, 32 activités et 2 ressources, 32 activités et 3 ressources, et 32 activités et 4 ressources;

- le retrait d'une série d'activités : nous avons construit 60 séries de problèmes. 10 séries de problèmes pour chacun des types de problèmes : 12 activités et 4 ressources, 22 activités et 4 ressources, 32 activités et une ressource, 32 activités et 2 ressources, 32 activités et 3 ressources, et 32 activités et 4 ressources;

- pour l'ajout d'une série de contraintes de précédence, 30 séries de probèmes ont été construites. 10 séries de problèmes comportant chacun 12 activités et 4 ressources, et 20 séries de problèmes comportant chacun 32 activités et entre 1 et 4 ressources. (les tests sur les problèmes comportant 22 activités sont en cours);

- le retrait d'une série de contraintes de précédence, nous avons construit 40 séries de problèmes. 10 séries de problèmes comportant chacun 12 activités et 4 ressources, 10 autres comportant chacun 22 activités et 4 ressources, et 20 séries de problèmes comportant chacun 32 activités et entre 1 et 4 ressources.

Dans ce qui suit, le symbole Na signifie que les valeurs utilisées sont obtenues par notre approche. Tandis que, le symbole Ro correspond au ré-ordonnancement.

Si nous notons par $T_{Na}(P_k)$ le temps (en secondes) nécessaire pour produire une solution du problème P_k par notre système, et $T_{Ro}(P_k)$ le temps nécessaire pour trouver une solution du problème P_k en utilisant l'approche classique (ré-ordonnancement), nous définissons le gain en temps par :

$$G_{temps}(P_i) = \frac{\sum_{0 \leq k \leq i} T_{Ro}(P_k) - \sum_{0 \leq k \leq i} T_{Na}(P_k)}{\sum_{0 \leq k \leq i} T_{Ro}(P_k)}$$

Pour chacune des mesures de performance définies précédemment, nous définissons le gain moyen en performance de notre approche par rapport au ré-ordonnancement par :

$$G_{mes}(P_i, P_j) = \frac{\sum_{1 \leq k \leq s} mes(S_i^k(Ro), S_j^k(Ro)) - \sum_{1 \leq k \leq s} mes(S_i^k(Na), S_j^k(Na))}{\sum_{1 \leq k \leq s} mes(S_i^k(Ro), S_j^k(Ro))}$$

où *mes* est une des mesures définies précédemment (DIFF, POSI, SDIF ou MDIF), s est le nombre des séries considérées, $S_i^k(Ro)$ est la solution du *ième* problème de la série k obtenue par le ré-ordonnancement, et enfin $S_i^k(Na)$ est la solution obtenue par notre approche.

Notons que lorsque le système n'admet pas de solution (en ajoutant une activité ou une contrainte de précédence, ou en modifiant une contrainte de précédence) la date de fin du projet (C_{max}) est augmentée automatiquement.

Nous précisons que tous les résultats présentés dans cette section sont détaillés dans l'annexe A.

10.4.4.1 Analyse des résultats en terme de temps CPU

Pour chaque aléa considéré et pour chaque classe de problèmes, nous produisons un tableau contenant les informations suivantes :

- **Min** qui correspond au gain minimal en temps sur l'ensemble des gains obtenus à une étape donnée de la série de problèmes.
- **Max** qui correspond au gain maximal en temps sur l'ensemble des gains obtenus à une étape donnée de la série de problèmes.
- **Moy** qui correspond à la moyenne des gains en temps de l'ensemble des gains obtenus à une étape donnée.
- **GMoy** qui correspond au gain entre la moyenne des temps nécessaires pour produire toutes les solutions successives de la série en utilisant notre approche et la moyenne des temps en utilisant l'approche classique.

Ajout d'une série d'activités : Le tableau 10.8 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une activité est ajoutée à chaque étape. La moyenne des gains est située entre 10.7% et 78.2%, et le gain $GMoy$ est très important, il se situe entre 43.7% et 90.5%. Nous pouvons constater un gain important sur tous les types de séries de problèmes. Ce gain devient très important sur les séries de problèmes comportant 32 activités.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		
	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	GMoy
12×4	-0.9%	37.7%	-63.8%	32.2%	–	–	–	–	43.7%
	64.4%		69.6%		–		–		
22×4	-34.6%	29.7%	-208%	5.8%	-155.2%	34.4%	–	–	78.9%
	94.4%		94.4%		96%		–		
32×1	30%	58.3%	18.2%	68.5%	40.7%	77%	-66.1%	46.7%	83.8%
	98.7%		99.3%		98.7%		99.6%		
32×2	-280.9%	10.7%	-140.1%	57.8%	-64.2%	68.7%	-78.9%	62.8%	84.9%
	91.2%		97.1%		96.9%		97.6%		
32×3	12.7%	61.4%	15.9%	66%	16.8%	74.6%	30.6%	78.2%	90.5%
	95.2%		98.2%		98.8%		96.3%		
32×4	33.3%	57%	-18.1%	63.5%	-19.6%	60.7%	26.7%	78.2%	90%
	75.9%		97.4%		98.7%		99.2%		

TAB. 10.8 – *Résultats pour le gain en temps pour des problèmes dans lesquels une série d'activités est ajoutée.*

Retrait d'une série d'activités : Dans le tableau 10.9, nous résumons les résultats obtenus pour la résolution des séries de problèmes dans lesquelles nous retirons une activité à chaque étape. La moyenne des gains est entre -51% et 65.5%, et le gain $GMoy$ est situé entre -19.1% et 59.4%. Nous constatons que pour les séries de problèmes comportant 12 activités, nous avons une perte globale. Alors que pour les autres séries de problèmes, nous avons un gain très important.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		
	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	GMoy
12×4	-212.2%	-51%	-195.5%	-65.3%	–	–	–	–	-19.1%
	21.7%		19.5%		–		–		
22×4	-637.8%	-22.9%	-242.8%	20.5%	-232.1%	21.5%	–	–	44.7%
	91.5%		94.3%		88.8%		–		
32×1	-185%	13.4%	-119%	20.6%	-102.6%	0.9%	-151.5%	-0.5%	47.7%
	97.1%		92.7%		91.1%		91.5%		
32×2	-148.4%	24.2%	-219.4%	14.4%	-161.2%	38.9%	-164.8%	34%	59.4%
	98%		98.7%		97.4%		97.5%		
32×3	0%	65.5%	-34.8%	52.3%	-38.3%	48.1%	-43.1%	49.2%	51.8%
	94.1%		94%		91.7%		90.3%		
32×4	-280.4%	31%	-259.5%	32.5%	-180.2%	28.3%	-148.1%	26.3%	59.4%
	92.4%		95.7%		92.3%		91.3%		

TAB. 10.9 – *Résultats pour le gain en temps pour des problèmes dans lesquels une série d'activités est retirée.*

Ajout d'une série de contraintes de précédence : Le tableau 10.10 résume les résultats que nous avons obtenus lors de la résolution des séries de problèmes dans lesquelles nous ajoutons à chaque étape une contrainte de précédence. La moyenne des gains se situe entre 18.7% et 41.4%, et le gain $GMoy$ se situe entre 15% et 25.4%. Nous pouvons constater que nous avons des gains importants sur les deux types de séries de problèmes.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		
	Min/Max	Moy.	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	GMoy
12 act.	-4.5%	28.1%	-47.5%	21.6%	-11.2%	30.1%	-43.6%	18.7%	25.4%
	60.1%		50.9%		63.9%		50.7%		
22 act.	-15.3%	30.3%	-0.4%	25.6%	–	–	–	–	28.5%
	90.9%		79.2%		–		–		
32 act.	-49.3%	41.4%	-58.1%	31.2%	-91.2%	23.7%	–	–	15%
	76.9%		85.2%		75.8%		–		

TAB. 10.10 – *Résultats pour le gain en temps pour des problèmes dans lesquels une série de contraintes de précédence est ajoutée.*

Retrait d'une série de contraintes de précédence : Le tableau 10.11 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles nous retirons une contrainte de précédence (en ajoutant sa négation) à chaque étape. La moyenne des gains se situe entre -22.1% et 43.6%, et le gain $GMoy$ est situé entre -32% et 70%. Nous pouvons constater que nous avons des pertes sur les séries de problèmes de taille 22, mais des gains importants sur les séries de problèmes de taille 32.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		
	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	GMoy
12 act.	-195.5%	4.3%	-148.8%	4.7%	–	–	–	–	70.5%
	96%		88.8%		–		–		
22 act.	-109.2%	2.3%	-64.7%	-2.8%	-152.1%	-22.1%	–	–	-32.5%
	47.9%		43.5%		37.8%		–		
32 act.	-329.4%	12.6%	-135.7%	33.6%	-142.1%	30.4%	-121.8%	43.6%	35.7%
	79.5%		86.6%		88.8%		89%		

TAB. 10.11 – *Résultats pour le gain en temps pour des problèmes dans lesquels une série de contraintes de précédence est retirée.*

10.4.4.2 Étude de la stabilité pour la mesure DIFF

Nous rappelons que cette mesure calcule le nombre d'activités dont la date de début a changé lors du passage d'une solution à l'autre.

Nous présentons, pour chaque aléa considéré, un tableau représentant le gain moyen en performance G_{DIFF} (*cf.* section 10.4 page 128).

Nous signalons que P_0 représente le problème initial, P_i le problème obtenu à la *iéme* perturbation. Et nous notons par P_{fin} le problème final.

Ajout d'une série d'activités : Dans le tableau 10.12, nous présentons les résultats obtenus lors de la résolution des séries de problèmes dans lesquelles une activité est ajoutée à chaque étape. Ces expérimentations montrent un gain très important pour les séries de problèmes comportant 32 activités : il se situe entre 0% et 76.2%. Nous pouvons constater un gain important lorsque nous comparons les solutions initiales et les solutions finales des séries de problèmes comportant 32 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-81.5%	-48%	–	–	-18.7%
22×4	8.3%	-2.9%	-10.9%	–	5%
32×1	61.7%	69.4%	71.4%	0%	-1.3%
32×2	54.7%	40.2%	33.6%	54.8%	21.9%
32×3	51.2%	50%	76.2%	20.8%	10%
32×4	57.8%	68.7%	38.4%	48.3%	23.7%

TAB. 10.12 – *Résultats pour le gain moyen en performance pour la mesure* DIFF *sur des problèmes dans lesquels une série d'activités est ajoutée.*

Retrait d'une série d'activités : Le tableau 10.13 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une activité est retirée à chaque étape. Ces résultats montrent un gain important pour les séries de problèmes comportant 22 et 32 activités. Nous constatons aussi un gain important entre les solutions initiales et les solutions finales pour tout les types de problèmes.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-19.5%	5.5%	–	–	12.7%
22×4	12.5%	17.5%	-3.2%	–	8.6%
32×1	16%	-0.8%	23.5%	8.3%	2.8%
32×2	23.9%	12.7%	15.2%	31.1%	20%
32×3	34.9%	20.3%	0.4%	36.9%	14.8%
32×4	18.1%	16.1%	18.2%	-16.9%	-0.3%

TAB. 10.13 – *Résultats pour le gain en performance pour la mesure* DIFF *sur des problèmes dans lesquels une série d'activités est retirée.*

Ajout d'une série de contraintes de précédence : Dans le tableau 10.14, nous résumons les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est ajoutée à chaque étape. D'après ces résultats, nous pouvons constater que nous avons des gains et des pertes. Mais, le gain entre les solutions initiales et les solutions finales est pratiquement nul.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	0%	7.5%	-19.3%	0%	0.1%
22	17.5%	15.2%	–	–	10.4%
32	9.9%	-1.8%	-7%	–	0%

TAB. 10.14 – *Résultats pour le gain en performance pour la mesure* DIFF *sur des problèmes dans lesquels une série de contraintes de précédence est ajoutée.*

Retrait d'une série de contraintes de précédence : Dans le tableau 10.15, nous résumons les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est retirée à chaque étape. Les résultats obtenus montrent que nous avons des pertes sur les séries de problèmes comportant 22 activtés et des gains assez importants sur les séries de problèmes comportant 32 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	-6.3%	-7.1%	–	–	0%
22	-11.7%	-22%	-15.3%	–	-3.6%
32	0%	17.7%	22.1%	-11.4%	10.6%

TAB. 10.15 – *Résultats pour le gain en performance pour la mesure* DIFF *sur des problèmes dans lesquels une série de contraintes de précédence est retirée.*

10.4.4.3 Étude de la stabilité pour la mesure POSI

Nous rappelons que la mesure POSI calcule le nombre de positions relatives qui ont changé lorsque nous passons d'une solution à l'autre.

Pour chaque aléa considéré, nous présentons un tableau qui contient le gain moyen en performance G_{POSI} présenté dans la section 10.4 (page 128).

Ajout d'une série d'activités : Le tableau 10.16 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une activité est ajoutée à chaque étape. La moyenne des gains est comprise entre -51.8% et 81%. Ces résultats montrent un gain important pour les séries de problèmes comportant 32 activités. Nous pouvons aussi constater un gain important entre les solutions initiales et les solutions finales pour cette même classe de séries de problèmes.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-24%	-51.8%	–	–	-58.3%
22×4	18.2%	12.5%	-14.6%	–	-19%
32×1	73.1%	79%	81%	-27.6%	11.2%
32×2	53.7%	37.4%	38.2%	63.5%	37.5%
32×3	54.3%	45.8%	76.5%	33.3%	21.1%
32×4	36.7%	62.1%	34.1%	49.7%	27.3%

TAB. 10.16 – *Résultats pour le gain en performance pour la mesure* POSI *sur des problèmes dans lesquels une série d'activités est ajoutée.*

Retrait d'une série d'activités : Le tableau 10.17 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une activité est retirée à chaque étape. La moyenne des gains est comprise entre -27% et 43.6%. Ces résultats montrent un gain important pour les séries de problèmes comportant 22 et 32 activités. Nous constatons aussi un gain important entre les solutions initiales et les solutions finales pour tout les types de problèmes.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-27%	1.7%	–	–	28.9%
22×4	17%	25.4%	-7.9%	–	25%
32×1	14.3%	7.2%	15.6%	30%	27.5%
32×2	35.7%	17.9%	-0.1%	39.1%	38.5%
32×3	-0.1%	27.7%	14.9%	43.6%	32.1%
32×4	12.9%	36.5%	20.2%	18.9%	0.9%

TAB. 10.17 – *Résultats pour le gain en performance pour la mesure* POSI *sur des problèmes dans lesquels une série d'activités est retirée.*

Ajout d'une série de contraintes de précédence : Dans le tableau 10.18, nous présentons les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est ajoutée à chaque étape. Ces résultats montrent un gain important pour les séries de problèmes comportant 32 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	0%	10.9%	-35.4%	-16.6%	-3.6%
22	31%	6.1%	–	–	9.2%
32	32.9%	28.6%	11.4%	–	16.2%

TAB. 10.18 – *Résultats pour le gain en performance pour la mesure* POSI *sur des problèmes dans lesquels une série de contraintes de précédence est ajoutée.*

Retrait d'une série de contraintes de précédence : Le tableau 10.19 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est retirée à chaque étape. D'après ces résultats, nous constatons un gain important pour les séries de problèmes comportant 32 activités, des pertes pour les autres types de séries. Nous constatons aussi que pour les séries de problèmes comportant 12 et 22 activités, le gain entre les solutions initiales et les solutions finales est pratiquement nul.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	-10.5%	-17.1%	–	–	0%
22	-4.1%	-26.1%	-28.4%	–	-0.3%
32	13.1%	25.3%	20.1%	19%	22.9%

TAB. 10.19 – *Résultats pour le gain en performance pour la mesure* POSI *sur des problèmes dans lesquels une série de contraintes de précédence est retirée.*

10.4.4.4 Étude de la stabilité pour la mesure SDIF

Nous rappelons que la mesure SDIF représente la somme des perturbations. En d'autres termes, c'est la somme des écarts entre les dates de début des activités dans les deux solutions.

Nous présentons, pour chacune des perturbations considérées, un tableau représentant le gain moyen en performance G_{SDIF} défini dans la section 10.4 (page 128).

Ajout d'une série d'activités : Le tableau 10.20 fournit les résultats obtenus lors de la résolution des séries de problèmes dans lesquelles une activité est ajoutée à chaque étape. Ces résultats montrent un gain très important pour les séries de problèmes comportant 32 activités. Nous constatons aussi un gain important entre les solutions initiales et les solutions finales pour les séries de problèmes qui comportent 32 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-60.6%	-69.3%	–	–	-21%
22×4	17.5%	20.7%	-19%	–	-5.6%
32×1	71.5%	77.6%	86.6%	-59.6%	-10.1%
32×2	56.4%	51.6%	39%	67.8%	47.2%
32×3	39.5%	45.2%	75.5%	30.5%	23.1%
32×4	32.1%	67.6%	23.4%	56%	19.7%

TAB. 10.20 – *Résultats pour le gain en performance pour la mesure* SDIF *sur des problèmes dans lesquels une série d'activités est ajoutée.*

Retrait d'une série d'activités : Dans le tableau 10.21, nous donnons les résultats obtenus concernant la résolution des séries de problèmes dans lesquelles une activité est retirée à chaque étape. Ces résultats montrent un gain important pour tout les types de problèmes. Nous constatons aussi un gain important entre les solutions initiales et les solutions finales.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-23.9%	13.2%	–	–	19.7%
22×4	15.4%	23.6%	-9.5%	–	20.6%
32×1	13.6%	7.9%	10.8%	11.4%	10.9%
32×2	46%	17.4%	10.9%	43.9%	44.6%
32×3	25.6%	20.4%	0.9%	33.9%	26.2%
32×4	16%	26.2%	25.6%	22.1%	6.9%

TAB. 10.21 – *Résultats pour le gain en performance pour la mesure* SDIF *sur des problèmes dans lesquels une série de activités est retirée.*

Ajout d'une série de contraintes de précédence : Le tableau 10.22 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est ajoutée à chaque étape. Ces résultats montrent un gain important pour les séries de problèmes comportant 32 activités. Nous constatons une perte pour les séries de problèmes comportant 12 activités, avec une perte moins importante entre les solutions initiales et les solutions finales.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	-3.3%	16.9%	-21.4%	-12.1%	-2.6%
22	31.9%	3.4%	–	–	9%
32	27.5%	27.2%	8.2%	–	17.9%

TAB. 10.22 – *Résultats pour le gain en performance pour la mesure* SDIF *sur des problèmes dans lesquels une série de contraintes de précédence est ajoutée.*

Retrait d'une série de contraintes de précédence : Dans le tableau 10.23, nous résumons les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est retirée à chaque étape. Ces résultats montrent un gain important pour les séries de problèmes comportant 32 activités. Nous constatons aussi un gain assez important entre les solutions initiales et les solutions finales pour tout les types de problèmes.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	5.9%	-9.8%	–	–	1.7%
22	-11%	-36.1%	-39.1%	–	2.5%
32	19.7%	18%	7.8%	19.9%	26%

TAB. 10.23 – *Résultats pour le gain en performance pour la mesure* SDIF *sur des problèmes dans lesquels une série de contraintes de précédence est retirée.*

10.4.4.5 Étude de la stabilité pour la mesure MDIF

La mesure MDIF représente la perturbation maximale, *i.e.* la différence entre les dates de début des activités, lorsque nous passons d'une solution à une autre.

Nous présentons, pour chacune des perturbations considérées, un tableau des gains moyens en performance G_{MDIF} associés à la mesure SDIF définie dans la section 10.4 (page 128).

Ajout d'une série d'activités : Le tableau 10.24 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une activité est ajoutée à chaque étape. Les résultats obtenus montrent un gain important pour les séries de problèmes comportant 32 activités. Nous constatons aussi un gain important entre les solutions initiales et les solutions finales pour les séries de problèmes de 32 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-41%	-48%	–	–	-28%
22×4	20.6%	22.6%	-15.2%	–	-18.2%
32×1	51.2%	74.6%	83.3%	-52.4%	-3.6%
32×2	43.3%	42.8%	16.6%	53.5%	21.9%
32×3	16.2%	42%	53%	39%	15.3%
32×4	32%	41.1%	24.1%	66%	15.5%

TAB. 10.24 – *Résultats pour le gain en performance pour la mesure* MDIF *sur des problèmes dans lesquels une série de contraintes de précédence est retirée.*

Retrait d'une série d'activités : Le tableau 10.25 présente les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une activité est retirée à chaque étape. Ces résultats montrent un gain important pour les séries de problèmes comportant 22 et 32 activités. Nous pouvons aussi constater un gain important entre les solutions initiales et les solutions finales pour tout les types de problèmes.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12×4	-0.9%	-7.3%	–	–	12.9%
22×4	20.4%	13.6%	-7.8%	–	16.6%
32×1	15.9%	5.9%	11.6%	17.1%	20.3%
32×2	41.5%	-0.5%	29.4%	36.3%	28.4%
32×3	17.7%	-0.4%	-0.3%	13.4%	21.2%
32×4	-10.8%	24.7%	8.2%	20.4%	21.4%

TAB. 10.25 – *Résultats pour le gain en performance pour la mesure* MDIF *sur des problèmes dans lesquels une série de contraintes de précédence est retirée.*

Ajout d'une série de contraintes de précédence : Le tableau 10.26 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte de précédence est ajoutée à chaque étape. Ces résultats montrent un gain important pour les séries de problèmes comportant 32 activités. Aussi, nous pouvons constater un gain important entre les solutions initiales et les solutions finales pour les séries de problèmes qui comportent 32 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	-0.1%	12.3%	-13.3%	-18.1%	-5.3%
22	23.2%	15.5%	–	–	-5.4%
32	10.4%	19.9%	9.1%	–	11.1%

TAB. 10.26 – *Résultats pour le gain en performance pour la mesure* MDIF *sur des problèmes dans lesquels une série de contraintes de précédence est ajoutée.*

Retrait d'une série de contraintes de précédence : Dans le tableau 10.27, nous résumons les résultats obtenus lors de la résolution des séries de problèmes dans lesquelles une contrainte de précédence est retirée à chaque étape. Les résultats obtenus montrent un gain important pour les séries de problèmes comportant 32 activités. Nous constatons aussi un gain important pour les séries de problèmes de 32 activités et quasiment nul pour les séries de problèmes de 12 activités.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_0 \rightarrow P_{fin}$
12	12.6%	-11.7%	–	–	0%
22	4, 6%	6%	-22.6%	–	2.7%
32	35.1%	7.3%	7.7%	20.5%	16.3%

TAB. 10.27 – *Résultats pour le gain en performance pour la mesure* MDIF *sur des problèmes dans lesquels une série de contraintes de précédence est retirée.*

10.4.4.6 Synthèse

Nous pouvons donner une synthèse des résultats obtenus en signalant que nous avons des gains très importants en temps et en stabilité pour tous les types de séries et pour toutes les perturbations considérées, sauf pour :

- les séries de problèmes de petite taille (comportant 12 activités) lorsque nous retirons une série d'activités. Les pertes en temps de calcul peuvent s'expliquer par le fait que ces problèmes sont très faciles à résoudre et deviennent de plus en plus faciles lorsque le nombre d'activités est réduit à chaque étape. Le temps de réparation dépasse donc le temps nécessaire pour le recalcul d'une nouvelle solution. Par contre, nous avons des gains importants en terme de stabilité, et cela pour les quatre mesures considérées;

- les séries de problèmes de taille moyenne (qui comportent 22 activités) lorsque nous retirons une série de contraintes de précédence (en ajoutant leur négation). L'ajout de la négation des contraintes de précédence perturbe la solution courante, notre approche passe donc plus de temps à réparer la solution courante. Les résultats obtenus sur la stabilité montrent bien que les solutions successives produites par notre système sont plus perturbées que celles produites par le ré-ordonnancement. Nous pouvons aussi constater que les solutions finales produites par notre approche restent proches des solutions initiales;

- les séries de problèmes de petite taille lorsque nous ajoutons une série d'activités. Nous constatons des gains importants en temps de calcul, mais des pertes en stabilité et ceci pour toutes les mesures. Sans doute car l'ajout des contraintes de précédence (reliant

les successeurs et les prédécesseurs de l'activité ajoutée) provoquent des perturbations sur les dates de début des activités. Et comme ces problèmes sont de petite taille, ces perturbations sont donc considérables.

10.5 Conclusion

Dans ce chapitre, nous avons réalisé différentes expérimentations sur des jeux de tests de la littérature pour la résolution du RCPSP dans le cadre statique, puis des tests sur des séries de problèmes que nous avons construits à partir de jeux de tests de la littérature pour le cadre dynamique.

Les résultats que nous avons obtenus, dans le cadre statique, montrent que notre approche est satisfaisante par rapport aux approches existantes.

Dans le cadre dynamique, nous avons comparé notre approche à l'approche classique (réordonnancement) en terme de temps de calcul CPU, puis en terme de stabilité. Nous avons introduit quatre mesures permettant de calculer les gains de notre approche par rapport à l'approche classique.

En terme de temps de calcul, les gains sont très importants sur des séries de problèmes de grande taille, moins importants sur des séries de problèmes de taille moyenne et nous constatons des pertes sur des séries de problèmes de petite taille.

En terme de stabilité, nous obtenons des gains très importants sur des séries de problèmes de grande taille pour les quatre mesures considérées. Les gains sont moins importants sur des séries de problèmes de taille moyenne pour quelques mesures, et nous constatons quelques pertes sur des séries de petite taille.

Troisième partie

Résolution de quelques extensions du RCPSP dynamique

Introduction

Dans la partie précédente, nous avons présenté notre approche pour la résolution du RCPSP dans le cadre dynamique. Cette partie sera consacrée à la résolution de quelques extensions du RCPSP dynamiques.

Cette partie est organisée comme suit :

- Le premier chapitre présente une généralisation des contraintes temporelles (précédence, chevauchement et disjonction) basée sur la notion de délai. Nous avons modélisé ces contraintes généralisées en contraintes simples, présenté leur mécanisme de propagation et calculé les explications associées.
- Le second chapitre présente une généralisation des contraintes de ressources développées pour le RCPSP dynamique. Nous présentons une généralisation des contraintes *parties obligatoire* et *intervalles de tâches* qui consiste à généraliser les règles de déduction utilisées dans ces contraintes.
- En fin, le dernier chapitre est consacré aux résultats expérimentaux.

Chapitre 11

Contraintes temporelles généralisées

Sommaire

Introduction

Dans le chapitre 8, nous nous sommes interessés aux contraintes temporelles simples, à savoir les contraintes de précédence, de parallélisme (ou de chevauchement), et de disjonction. Or ces contraintes ne suffisent souvent pas pour modéliser les relations temporelles entre activités apparaissant dans certains problèmes d'ordonnancement réels. En effet, les contraintes temporelles imposées par ces problèmes sont plus complexes.

Ainsi, pour le RCPSP, plusieurs extensions peuvent être considérées. Citons par exemple, le RCPSP avec des contraintes de précédence généralisées ou le RCPSP généralisé (*cf.* section 2.6 page 21). Ces extensions sont peu étudiées dans la littérature par rapport au RCPSP classique. Sans doute car l'integration des contraintes temporelles, imposées par ces problèmes, dans les techniques utilisées en recherche opérationnelle est plus délicate.

Dans ce chapitre, nous définissons une généralisation des contraintes de précédence, de chevauchement et de disjonction. Nous en présentons une modélisation basée sur la notion de distance (*cf.* section 7.2 page 71). En suite, nous présentons le mécanisme de propagation de ces contraintes, et le calcul des explications associées.

11.1 Généralisation des contraintes temporelles : précédence, parallélisme et disjonction

Pour les problèmes classiques d'ordonnancement, les contraintes temporelles rencontrées sont principalement les contraintes de précédence, les contraintes disjonctives, ou les contraintes de précédence généralisées (*cf.* section 2.6 page 21). Cependant, ces contraintes restent insuffisantes, car les contraintes temporelles que nous pouvons rencontrer dans des problèmes réels sont beaucoup plus complexes.

Les contraintes temporelles simples peuvent être généralisées en prenant en compte les délais séparant les dates de début et de fin des activités. Ces délais peuvent être des durées constantes d, ou des durées bornées par deux valeurs d_{min} et d_{max}.

11.1.1 Généralisation des contraintes de précédence

Nous pouvons définir deux types de contraintes :

- la contrainte qui impose à une activité j de commencer exactement d unités de temps après la fin d'une activité i. Cette contrainte est notée $i \rightarrow^{d} j$ (*Fig.* 11.1 (a)).
- la contrainte imposant à une activité j de ne pas démarrer avant que d_{min} unités de temps se soient écoulées après la date de fin de l'activité i, et après que d_{max} unités de temps se soient écoulées après la date de fin de l'activité i. Cette contrainte est notée $i \rightarrow_{d_{min}}^{d_{max}} j$ (*Fig.* 11.1 (b)).

11.1.2 Généralisation des contraintes de chevauchement

Les contraintes que nous considérons sont :

- la contrainte imposant à une activité j de commencer après la date de début d'une activité i de telle sorte que la durée de chevauchement entre ces deux activités soit exactement d unités de temps. Cette contrainte est notée $i \parallel^{d} j$ (*Fig.* 11.2 (a)).

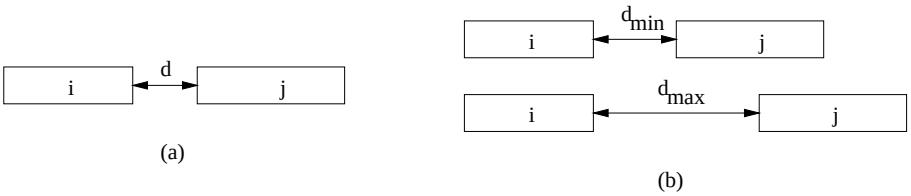


FIG. 11.1 – *Contraintes de précédence généralisées.*

– la contrainte qui impose à une activité j de commencer après la date de début d'une activité i de telle sorte que la durée de chevauchement entre ces deux activités soit au moins d_{min} et au plus d_{max} unités de temps. Cette contrainte est notée $i \|_{d_{min}}^{d_{max}} j$ (*Fig.* 11.2 (b)).

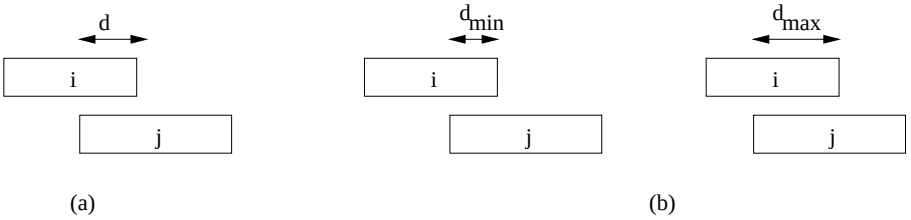


FIG. 11.2 – *Contraintes de chevauchement généralisées.*

11.1.3 Généralisation des contraintes disjonctives

Nous considérons les contraintes suivantes :

– la contrainte qui impose à deux activités i et j d'être en disjonction de telle sorte qu'il y ait exactement d unités de temps qui séparent la date de fin de l'une des activités et la date de début de l'autre. Cette contrainte est notée $i \leftrightarrow^{d} j$ (*Fig.* 11.3 (a)).

– la contrainte imposant à deux activités i et j d'être en disjonction de telle sorte que la durée qui sépare la date de fin de l'une des activités et la date de début de l'autre soit comprise entre d_{min} et d_{max} unités de temps. Cette contrainte est notée $i \leftrightarrow_{d_{min}}^{d_{max}} j$ (*Fig.* 11.3 (b)).

11.1.4 Quelques remarques

Les propriétés suivantes découlent des définitions présentées précédement pour les contraintes temporelles simples généralisées :

– la contrainte $i \rightarrow_{d}^{d} j$ est équivalente à la contrainte $i \rightarrow^{d} j$;

– les deux contraintes $i \leftrightarrow_{d}^{d} j$ et $i \leftrightarrow^{d} j$ sont équivalentes;

– la contrainte $i \|_{d}^{d} j$ est équivalente à la contrainte $i \|^{d} j$;

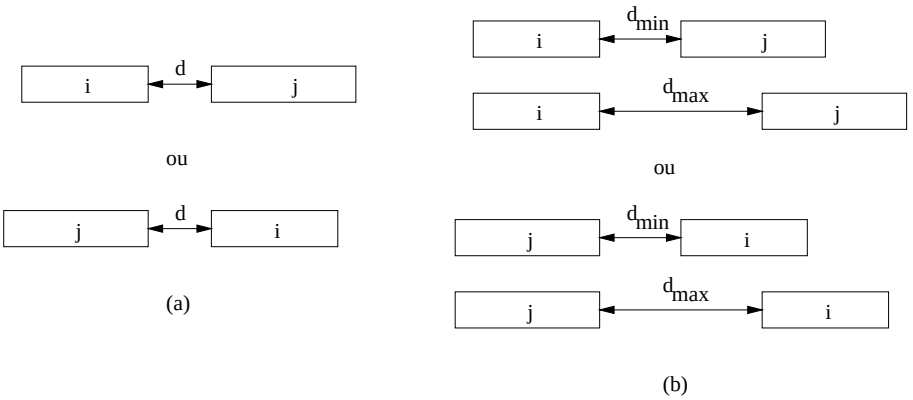


FIG. 11.3 – *Contraintes disjonctives généralisées.*

- les deux contraintes $i \|^d j$ et $j \|^d i$ sont différentes;
- la contrainte $i \|_{d_{min}}^{d_{max}} j$ n'est pas équivalente à $j \|_{d_{min}}^{d_{max}} i$;
- la contrainte $i \|^d j$ n'est valide que si le délai d est compris entre 1 et $\min\{p_i, p_j\}$;
- la contrainte $i \|_{d_{min}}^{d_{max}} j$ n'est valide que si les délais d_{min} et d_{max} sont compris entre 1 et $\min\{p_i, p_j\}$.

11.1.5 Un exemple illustratif

Afin d'illustrer le mécanisme de propagation des contraintes temporelles simples généralisées ainsi que le calcul des explications associées, l'exemple suivant est considéré dans la suite de ce chapitre.

> **Exemple 40 (Un exemple) :**
> Considérons un problème d'ordonnancement comportant six activités ayant chacune une durée opératoire p_i, une date de début au plus tôt r_i et une date au plus tard f_i. Soit $UB = 12$ une borne supérieure de la date de fin de l'ordonnancement. Au début tous les r_i sont nuls et les f_i sont égaux à $UB - p_i$. Le tableau 11.1 fournit les données du problème.
> Ces activités sont liées par les contraintes temporelles généralisées suivantes : $1 \rightarrow^2 2$, $1 \rightarrow_3^6 5$, $2 \|^1 6$, $1 \|_1^3 3$, $3 \leftrightarrow^2 6$, et $3 \leftrightarrow_2^3 4$.

La figure 11.4 représente les contraintes temporelles liant les six activités du problème. Et la figure 11.5 représente une solution réalisable du problème. Dans cette figure, nous représentons aussi les délais entre les activités liées par les contraintes temporelles du problème. Ainsi, par exemple le délai de chevauchement entre les deux activités 1 et 3 est égal à 2 (la contrainte $1 \|_1^3 3$ est bien satisfaite). De même, le délai entre le début de l'activité 4 et la fin de l'activité 3 est égal à 2. Ce qui satisfait bien la contraite $3 \leftrightarrow_2^3 4$.

Activité	p_i	r_i	f_i
1	4	0	8
2	3	0	9
3	4	0	8
4	3	0	9
5	1	0	11
6	4	0	8

TAB. 11.1 – *Données de l'exemple 40*

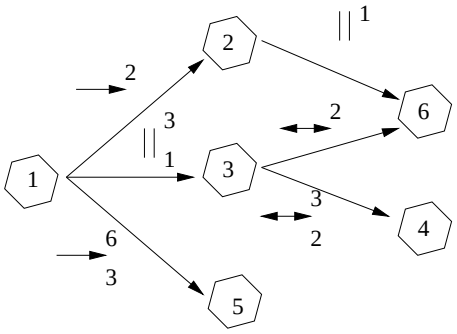


FIG. 11.4 – *Le graphe des contraintes temporelles généralisées.*

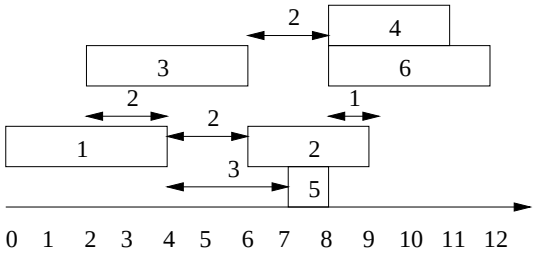


FIG. 11.5 – *Une solution réalisable du problème.*

11.2 Modélisation des contraintes temporelles

Dans notre système, nous avons modélisé les contraintes temporelles généralisées en contraintes simples. La modélisation de ces contraintes temporelles est basée sur la notion de distance (*cf.* section 7.2 page 71). Le tableau 11.2 fournit la modélisation pour chacune de ces contraintes temporelles.

Contrainte	Modélisation
$i \rightarrow^{d} j$	$d_{ij} = d$
$i \rightarrow_{d_{min}}^{d_{max}} j$	$d_{min} \leq d_{ij} \leq d_{max}$
$i \leftrightarrow^{d} j$	$d_{ij} = d \vee d_{ji} = d$
$i \leftrightarrow_{d_{min}}^{d_{max}} j$	$(d_{min} \leq d_{ij} \leq d_{max}) \vee (d_{min} \leq d_{ji} \leq d_{max})$
$i \parallel^{d} j$	$d_{ij} = -d$
$i \parallel_{d_{min}}^{d_{max}} j$	$-d_{max} \leq d_{ij} \leq -d_{min}$

TAB. 11.2 – *Modélisation des contraintes temporelles généralisées*

Exemple 41 (Modélisation) :
Le CSP correspondant à l'exemple 40 est défini par :
Les variables : $t_1, t_2, t_3, t_4, t_5, t_6, d_{12}, d_{15}, d_{51}, d_{26}, d_{13}, d_{31}, d_{36}, d_{63}, d_{34}, d_{43}$.
Les domaines : $D_1 = [0,8]$, $D_2 = [0,9]$, $D_3 = [0,8]$, $D_4 = [0,9]$, $D_5 = [0,11]$, $D_6 = [0,8]$, $D_{12} = [-12,5]$, $D_{15} = [-12,7]$, $D_{51} = [-12,7]$, $D_{26} = [-12,6]$, $D_{13} = [-12,4]$, $D_{31} = [-12,4]$, $D_{36} = [-12,4]$, $D_{63} = [-12,4]$, $D_{34} = [-12,6]$, $D_{43} = [-12,6]$.
Les contraintes : $c_1 : d_{12} = 2$, $c_2 : 3 \leq d_{15} \leq 6$, $c_3 : d_{26} = -1$, $c_4 : -3 \leq d_{13} \leq -1$, $c_5 : (d_{36} = 2 \vee d_{63} = 2)$, $c_6 : (2 \leq d_{34} \leq 3) \vee (2 \leq d_{43} \leq 3)$.
Notons qu'à chaque fois que nous créons une variable d_{ij}, la *contrainte de distance* $d_{ij} = t_j - t_i - p_i$ est automatiquement ajoutée au réseau de contraintes.

Dans la figure 11.6, nous donnons une représentation des activités du problème.

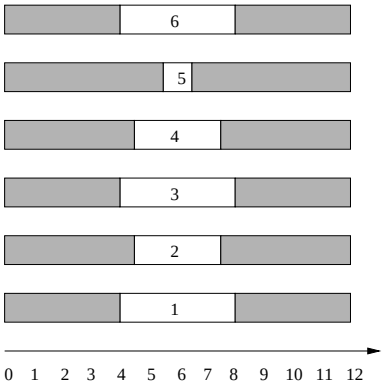


FIG. 11.6 – *Représentation des activités du problème de l'exemple 41.*

Algorithme 22: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $i \rightarrow^{d_{max}}_{d_{min}} j$ et v représente une de ses variables

awakeOnInf(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1 $updateInf(d_{ji}, -d_{min} - p_i - p_j)$
  si $v = d_{ji}$ **alors**
2 $updateSup(d_{ij}, d_{max})$
fin

11.3 Propagation des contraintes temporelles

Comme dans notre système, la réduction des domaines se fait par modification des bornes des domaines, le filtrage se fait donc par les deux méthodes : `awakeOnInf` et `awakeOnSup` (*cf.* section 5.1.4 page 50).

11.3.1 Contraintes de précédence généralisées

11.3.1.1 La contrainte $i \rightarrow^{d} j$

Cette contrainte est modélisée par la contrainte unaire $d_{ij} = d$ qui est une des contraintes de base du système CHOCO. Le domaine D_{ij} de la variable d_{ij} est réduit au singleton $\{d\}$.

11.3.1.2 La contrainte $i \rightarrow^{d_{max}}_{d_{min}} j$

Cette contrainte est équivalente à la contrainte $j \rightarrow^{-d_{min}-p_i-p_j}_{-d_{max}-p_i-p_j} i$ (car $d_{ji} = -d_{ij} - p_i - p_j$). Nous avons introduit cette contrainte comme une contrainte à deux variables d_{ij} et d_{ji}.

Le principe de propagation de cette contrainte est comme suit :

- si la borne inférieure de la variable d_{ij} augmente alors la borne inférieure de la variable d_{ji} est ajustée à la valeur $-d_{max} - p_i - p_j$ (ligne 1 de l'algorithme 22). Par contre, lorsque la borne inférieure de la variable d_{ji} augmente alors la borne supérieure de la variable d_{ij} est ajustée à la valeur d_{max} (ligne 2 de l'algorithme 22).

- si la borne supérieure de la variable d_{ij} diminue alors la borne supérieure de la variable d_{ji} est ajustée à la valeur $-d_{min} - p_i - p_j$ (ligne 1 de l'algorithme 23). Par contre, si la borne supérieure de la variable d_{ji} diminue alors la borne inférieure de la variable d_{ij} est ajustée à la valeur d_{min} (ligne 2 de l'algorithme 23).

Cette procédure nous permet d'ajuster les bornes des deux variables d_{ij} et d_{ji}. En effet, si la borne inférieure de la variable d_{ij} augmente alors la borne inférieure de la variable d_{ji} augmente aussi, ce qui va causer la réduction de la borne supérieure de la variable d_{ij} et par suite la diminution de la borne supérieure de la variable d_{ji}, et enfin la réduction de la borne inférieure de la variable d_{ij}.

Algorithme 23: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $i \rightarrow_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1 $updateSup(d_{ji}, -d_{max} - p_i - p_j)$
  si $v = d_{ji}$ **alors**
2 $updateInf(d_{ij}, d_{min})$
fin

Exemple 42 (Contraintes de précédence généralisées) :
Considérons le CSP définie dans l'exemple 41.
La propagation de la contrainte $c_1 : d_{12} = 2$ transforme le domaine $D_{12} = [-12, 5]$ en $D_{12} = \{2\}$, et par suite la contrainte de distance $C_{12} : d_{12} = t_2 - t_1 - p_1$ réduit respectivement les domaines des variables t_1 et t_2 à $[0, 3]$ et $[6, 9]$.
Grâce à la contrainte $d_{15} = t_5 - t_1 - p_1$, le domaine de la variable d_{15} est réduit à $[-7, 5]$. Ce qui déclanche la propagation de la contrainte $c_2 : 3 \leq d_{15} \leq 6$, le domaine de d_{15} sera donc réduit à $[3, 5]$, et celui de d_{51} à $[-10, -8]$.
Dans la figure 11.7, nous donnons une représentation des activités dont les domaines ont été réduit par propagation des deux contraintes $1 \rightarrow^2 2$ et $1 \rightarrow_3^6 5$.

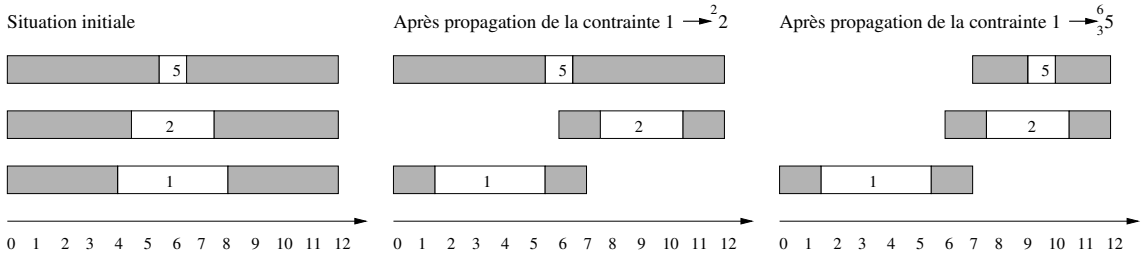


FIG. 11.7 – *Représentation des activités après propagation des contraintes de précédence généralisées.*

Remarque : Nous avons choisi d'introduire la contrainte $i \rightarrow_{d_{min}}^{d_{max}} j$ comme une contrainte à deux variables dans le but d'accélérer la propagation. En effet, supposons que la distance d_{ji} est déjà introduite dans le système, la contrainte de distance $C_{ji} : d_{ji} = t_i - t_j - p_j$ fait alors partie du réseau de contraintes. L'augmentation de la borne inférieure (par exemple) de la variable d_{ji} n'entraîne pas forcement la réduction de l'une des bornes des variables t_i et t_j (*cf.* l'exemple 43). Ce qui n'aura donc aucune influence sur le domaine de la variable d_{ij} si nous ne considérons que la contrainte de distance $C_{ij} : d_{ij} = t_j - t_i - p_i$.

Algorithme 24: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $i \|_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnInf(C: **contrainte**, v: **distance**)
début
  si $v = d_{ij}$ **alors**
1 $updateInf(d_{ji}, d_{min} - p_i - p_j)$
  si $v = d_{ji}$ **alors**
2 $updateSup(d_{ij}, -d_{min})$
fin

Exemple 43 (Contre exemple) :
Considérons l'exemple suivant : $D_{ji} = [-6, 4]$, $D_i = [1, 5]$, $D_j = [0, 6]$ et $p_j = 1$ et supposons que le domaine D_{ji} est réduit à $[-4, 4]$. Comme les deux bornes (1 et 5) du domaine D_i ont au mois un support (4 et 0) dans D_j, et les deux bornes (0 et 6) du domaine D_j ont au moins un support (5 et 3) dans D_i, les domaines des variables t_i et t_j ne seront pas réduits.

11.3.2 Contraintes de chevauchement généralisées

Deux activités i et j vérifient la relation généralisée $i \|^d j$ si et seulement si $d_{ij} = -d$, et la relation $i \|_{d_{min}}^{d_{max}} j$ si et seulement si $-d_{max} \leq d_{ij} \leq -d_{min}$.

11.3.2.1 La contrainte $i \|^d j$

Cette contrainte modélisée par la contrainte unaire $d_{ij} = -d$ existe déjà dans le système CHOCO. Le domaine D_{ij} de la variable d_{ij} sera réduit au singleton $\{-d\}$.

11.3.2.2 La contrainte $i \|_{d_{min}}^{d_{max}} j$

Le principe de la propagation de cette contrainte suit le même principe que la contrainte de précédence généralisée $i \rightarrow_{d_{min}}^{d_{max}} j$ (*cf.* section 11.3.1.2).

- si la borne inférieure de la variable d_{ij} augmente alors la borne inférieure de la variable d_{ji} est ajustée à la valeur $d_{min} - p_i - p_j$ (ligne 1 de l'algorithme 24). Par contre, lorsque la borne inférieure de la variable d_{ji} augmente alors la borne supérieure de la variable d_{ij} est ajustée à la valeur $-d_{min}$ (ligne 2 de l'algorithme 24).

- si la borne supérieure de la variable d_{ij} diminue alors la borne supérieure de la variable d_{ji} est ajustée à la valeur $d_{max} - p_i - p_j$ (ligne 1 de l'algorithme 25). Par contre, si la borne supérieure de la variable d_{ji} diminue alors la borne inférieure de la variable d_{ij} est ajustée à la valeur $-d_{max}$ (ligne 2 de l'algorithme 25).

Algorithme 25: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $i \|_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1 $updateSup(d_{ji}, d_{max} - p_i - p_j)$
  si $v = d_{ji}$ **alors**
2 $updateInf(d_{ij}, -d_{max})$
fin

Exemple 44 (Contraintes de chevauchement généralisées) :
Reprenons l'exemple 41 page 156. La propagation de la contrainte $c_3 : d_{26} = -1$ transforme le domaine $D_{26} = [-12, 6]$ en $D_{26} = \{-1\}$, et par suite la contrainte de distance $d_{26} = t_6 - t_2 - p_2$ réduit respectivement les domaines des variables t_2 et t_6 à $\{6\}$ et $\{8\}$.
Comme le domaine de la variable t_1 a été réduit à $\{0\}$ (à cause de la contrainte de distance $d_{12} = t_2 - t_1 - p_1$), le domaine de la variable d_{31} a été réduit (à cause de la contrainte de distance $d_{31} = t_1 - t_3 - p_3$) et la contrainte $c_3 : \ -3 \leq d_{13} \leq -1$ est réveillée. Les domaines des variables d_{13} et d_{31} sont alors réduits respectivement à $[-3, -1]$ et $[-7, -5]$. Par conséquent, à cause de la contrainte de distance $d_{13} = t_3 - t_1 - p_1$, le domaine de la variable t_3 sera réduit à l'intervalle $[1, 3]$.
Dans la figure 11.8, nous représentons les activités dont les domaines ont été réduits par propagation des deux contraintes $2 \|^1 6$ et $1 \|_1^3 3$.

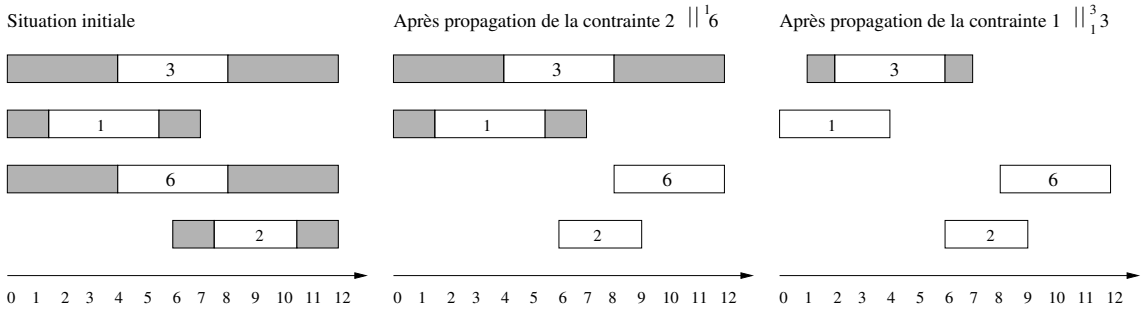


FIG. 11.8 – *Représentation des activités après propagation des contraintes de chevauchement généralisées.*

11.3.3 Contraintes disjonctives généralisées

Comme nous l'avons vu précédemment, deux activités i et j vérifient la relation généralisée $i \leftrightarrow^d j$, si et seulement si leurs distances d_{ij} et d_{ji} vérifient la contrainte $d_{ij} = d \vee d_{ji} = d$.

Algorithme 26: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $(d_{ij} = d \vee d_{ji} = d)$, et v est une de ses variables

awakeOnInf(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1   **si** $d_{ij}^{min} > d$ **alors**
      $updateInf(d_{ji}, d)$
      $updateSup(d_{ji}, d)$
  si $v = d_{ji}$ **alors**
2   **si** $d_{ji}^{min} > d$ **alors**
      $updateInf(d_{ij}, d)$
      $updateSup(d_{ij}, d)$
fin

De même, ces deux activités vérifient la relation généralisée $i \leftrightarrow_{d_{min}}^{d_{max}} j$, si et seulement si les distances d_{ij} et d_{ji} vérifient la contrainte $(d_{min} \leq d_{ij} \leq d_{max}) \vee (d_{min} \leq d_{ji} \leq d_{max})$.

11.3.3.1 La contrainte $i \leftrightarrow^d j$

Le principe de propagation de cette contrainte est comme suit :

- si la borne inférieure de la variable d_{ij} devient strictement supérieure à d alors le domaine de la variable d_{ji} est réduit au singleton $\{d\}$ (ligne 1 de l'algorithme 26). Réciproquement, lorsque la borne inférieure de la variable d_{ji} devient strictement supérieure à d alors le domaine de d_{ij} est réduit au singleton $\{d\}$ (ligne 2 de l'algorithme 26).

- si la borne supérieure de la variable d_{ij} devient strictement inférieure à d alors le domaine de la variable d_{ji} sera réduit au singleton $\{d\}$ (ligne 1 de l'algorithme 27). Réciproquement, lorsque la borne supérieure de d_{ji} devient strictement inférieure à d alors le domaine de la variable d_{ij} sera réduit au singleton $\{d\}$ (ligne 2 de l'algorithme 27).

Remarque : Signalons que lorsque la borne inférieure de la variable d_{ij} devient strictement supérieure à d, la contrainte $i \leftrightarrow^d j$ n'est plus satisfaite, nous avons donc une contradiction. La propagation de cette contrainte détectera cette contradiction lorsque le domaine de la variable d_{ji} sera réduit au singleton $\{d\}$. En effet, avant la propagation le domaine de la variable d_{ji} ne contenait que des valeurs négatives $(d_{ji} = -d_{ij} - p_i - p_j)$.

11.3.3.2 La contrainte $i \leftrightarrow_{d_{min}}^{d_{max}} j$

Le propagation de cette contrainte est comme suit :

- si la borne inférieure de la variable d_{ij} est strictement supérieure à d_{max} alors la borne inférieure de la variable d_{ji} devient d_{min} et la borne supérieure devient d_{max} (ligne 1 de l'algorithme 28). Réciproquement, lorsque la borne inférieure de la variable d_{ji} devient strictement supérieure à d_{max} alors la borne inférieure de d_{ij} devient d_{min} et la borne supérieure devient d_{max} (ligne 2 de l'algorithme 28).

Algorithme 27: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $(d_{ij} = d \vee d_{ji} = d)$, et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1   **si** $d_{ij}^{max} < d$ **alors**
      $updateInf(d_{ji}, d)$
      $updateSup(d_{ji}, d)$
  si $v = d_{ji}$ **alors**
2   **si** $d_{ji}^{max} < d$ **alors**
      $updateInf(d_{ij}, d)$
      $updateSup(d_{ij}, d)$
fin

Algorithme 28: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $(d_{min} \leq d_{ij} \leq d_{max} \vee d_{min} \leq d_{ji} \leq d_{max})$, et v représente une de ses variables

awakeOnInf(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1   **si** $d_{ij}^{min} > d_{max}$ **alors**
      $updateInf(d_{ji}, d_{min})$
      $updateSup(d_{ji}, d_{max})$
  si $v = d_{ji}$ **alors**
2   **si** $d_{ji}^{min} > d_{max}$ **alors**
      $updateInf(d_{ij}, d_{min})$
      $updateSup(d_{ij}, d_{max})$
fin

– si la borne supérieure de la variable d_{ij} est strictement inférieure à d_{min} alors la borne inférieure de la variable d_{ji} devient d_{min} et la borne supérieure devient d_{max} (ligne 1 de l'algorithme 29). Réciproquement, lorsque la borne supérieure de d_{ji} est strictement inférieure à d_{min} alors la borne inférieure de la variable d_{ij} devient d_{min} et la borne supérieure devient d_{max} (ligne 2 de l'algorithme 29).

Remarque : Signalons que lorsque la borne inférieure de la variable d_{ij} devient strictement supérieure à d_{max}, nous avons une contradiction car la contrainte $i \leftrightarrow_{d_{min}}^{d_{max}} j$ n'est pas satisfaite. Cette contradiction sera détectée lors de la propagation de cette contrainte. En effet, le domaine de la variable d_{ji} doit être réduit à l'intervalle $[d_{min}, d_{max}]$. Et comme son domaine D_{ji} ne contenait que des valeurs négatives $(d_{ji} = -d_{ij} - p_i - p_j)$, D_{ji} deviendra donc vide.

Algorithme 29: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $(d_{min} \leq d_{ij} \leq d_{max} \vee d_{min} \leq d_{ji} \leq d_{max})$, et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1   **si** $d_{ij}^{max} < d_{min}$ **alors**
      $updateInf(d_{ji}, d_{min})$
      $updateSup(d_{ji}, d_{max})$
  si $v = d_{ji}$ **alors**
2   **si** $d_{ji}^{max} < d_{min}$ **alors**
      $updateInf(d_{ij}, d_{min})$
      $updateSup(d_{ij}, d_{max})$
fin

Exemple 45 (Contraintes disjonctives généralisées) :
Considérons l'exemple 41 page 156. Comme les domaines des variables t_3 et t_6 sont devenus $[1,3]$ et $\{8\}$, la propagation de la contrainte de distance $d_{36} = t_6 - t_3 - p_3$ réduit le domaine de la variable d_{36} à $[1,3]$. De même la contrainte de distance $d_{63} = t_3 - t_6 - p_6$ réduit le domaine de la variable d_{63} à $[-11,-9]$. Par conséquent, la propagation de la contrainte c_5 : $(d_{36} = 2 \vee d_{63} = 2)$ réduit le domaine de la variable d_{36} à $\{2\}$ et celui de d_{63} à $\{-10\}$. Et donc grâce à la contrainte de distance $d_{36} = t_6 - t_3 - p_3$, le domaine de t_3 sera le singleton $\{2\}$.
Comme le domaine de la variable t_3 est devenu $\{2\}$, la propagation des contraintes $d_{34} = t_4 - t_3 - p_3$ et $d_{43} = t_3 - t_4 - p_4$ a réduit respectivement les domaines de d_{34} et d_{43} en $[-6,3]$ et $[-10,-1]$. Par conséquent, la propagation de la contrainte c_6 : $(2 \leq d_{34} \leq 3) \vee (2 \leq d_{43} \leq 3)$ met le domaine de d_{34} à $[2,3]$ et par suite le domaine de la variable t_4 sera réduit à $[8,9]$.
Dans la figure 11.9, nous donnons une représentation des activités dont les domaines ont été réduits par propagation des deux contraintes $3 \leftrightarrow^2 6$ et $3 \leftrightarrow_2^3 4$.

11.4 Calcul des explications

L'introduction des explications dans les contraintes temporelles généralisées est relativement aisé : il suffit d'ajouter de nouvelles informations dans les méthodes `awakeOnInf` et `awakeOnSup`. Les outils permettant d'enregistrer les explications pour chaque événement sont fournis dans le système PaLM (*cf.* section 5.1.4 page 50).

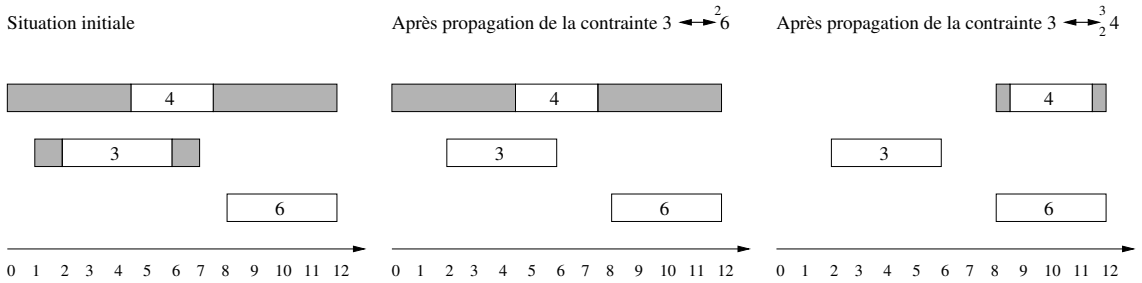


FIG. 11.9 – *Représentation des activités après propagation des contraintes de disjonction généralisées.*

11.4.1 Contraintes de précédence généralisées

11.4.1.1 La contrainte $i \rightarrow^{d} j$

Comme cette contrainte est une des contraintes de base du système CHOCO, elle l'est aussi pour le système PALM.

11.4.1.2 La contrainte $i \rightarrow_{d_{min}}^{d_{max}} j$

Pour la méthode `awakeOnInf`, la modification de la borne inférieure de la variable d_{ji} est due :

- à la contrainte elle-même. Cette contrainte doit être ajoutée à l'explication calculée. Le méthode `theConstraint` est alors appliquée (ligne 1 dans l'algorithme 30).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode $theInf(d_{ij})$ (ligne 1 dans l'algorithme 30).

Pour la méthode `awakeOnSup`, la modification de la borne supérieure de la variable d_{ji} est due :

- à la contrainte elle-même. Cette contrainte est ajoutée à l'explication calculée grâce à la méthode `theConstraint` (ligne 1 dans l'algorithme 31).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode $theInf(d_{ij})$ (ligne 1 dans l'algorithme 31).

Exemple 46 (Contraintes de précédence généralisées avec explication) :
Dans l'exemple 42 (page 157), la propagation de la contrainte c_2 : $1 \rightarrow_{3}^{d_6} 5$ réduit la borne inférieure de d_{15} de -7 à 3. L'explication de cette réduction est :

$$theInf(d_{51}) \wedge c_2$$

Algorithme 30: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $i \rightarrow_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnInf(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1 $updateInf(d_{ji},\ -d_{max} - p_i - p_j,\ becauseOf(theConstraint(C),\ theInf(d_{ij})))$
  si $v = d_{ji}$ **alors**
   $updateSup(d_{ij},\ d_{max},\ becauseOf(theConstraint(C),\ theInf(d_{ji})))$
fin

Algorithme 31: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $i \rightarrow_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1 $updateSup(d_{ji},\ -d_{min} - p_i - p_j,\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
  si $v = d_{ji}$ **alors**
   $updateInf(d_{ij},\ d_{min},\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
fin

11.4.2 Contraintes de chevauchement généralisées

11.4.2.1 La contrainte $i \|^d j$

La contrainte $d_{ij} = -d$ a été déjà introduite dans CHOCO, elle l'est aussi dans le système PaLM.

11.4.2.2 La contrainte $i \|_{d_{min}}^{d_{max}} j$

Pour la méthode `awakeOnInf`, la modification de la borne inférieure de la variable d_{ji} est due :

- à la contrainte elle-même. Cette contrainte doit être ajoutée à l'explication calculée. Le méthode `theConstraint` est alors appliquée (ligne 1 dans l'algorithme 32).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode $theInf(d_{ij})$ (ligne 1 dans l'algorithme 32).

Pour la méthode `awakeOnSup`, la modification de la borne supérieure de la variable d_{ji} est due :

- à la contrainte elle-même. Cette contrainte est ajoutée à l'explication calculée grâce à la méthode `theConstraint` (ligne 1 dans l'algorithme 33).

Algorithme 32: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $i \|_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnInf(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
1   $updateInf(d_{ji},\ d_{min} - p_i - p_j,\ becauseOf(theConstraint(C),\ theInf(d_{ij})))$
  si $v = d_{ji}$ **alors**
    $updateSup(d_{ij},\ -d_{min},\ becauseOf(theConstraint(C),\ theInf(d_{ji})))$
fin

Algorithme 33: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $i \|_{d_{min}}^{d_{max}} j$ et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
    $updateSup(d_{ji},\ d_{max} - p_i - p_j,\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
  si $v = d_{ji}$ **alors**
2   $updateInf(d_{ij},\ -d_{max},\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
fin

– aux contraintes responsables de la modification de la borne supérieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode $theSup(d_{ij})$ (ligne 2 dans l'algorithme 33).

Exemple 47 (Contraintes de chevauchement généralisées avec explication) :
Reprenons l'exemple 44 page 159. La propagation de la contrainte $c_3 : -3 \leq d_{13} \leq -1$ réduit le domaine de la variable d_{13} à $D_{13} = [-3, -1]$.
- Pour la réduction de la borne inférieure l'explication associée est :

$$theSup(d_{31}) \wedge c_3$$

- Pour la réduction de la borne supérieure l'explication associée est :

$$theInf(d_{31}) \wedge c_3$$

Algorithme 34: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

```
C est une contrainte de type i ↔^d j, et v est une de ses variables
awakeOnInf(C:contrainte, v:distance)
début
    si v = d_ij alors
        si d_ij^min > d alors
1           updateInf(d_ji, d, becauseOf(theConstraint(C), theInf(d_ij)))
            updateSup(d_ji, d, becauseOf(theConstraint(C), theInf(d_ij)))
    si v = d_ji alors
        si d_ji^min > d alors
            Soit e := Explanation() dans
                updateInf(d_ij, d, becauseOf(theConstraint(C), theInf(d_ji)))
                updateSup(d_ij, d, becauseOf(theConstraint(C), theInf(d_ji)))
fin
```

11.4.3 Contraintes disjonctives généralisées

11.4.3.1 La contrainte $i \leftrightarrow^d j$

Pour la méthode `awakeOnInf`, la modification de la borne inférieure de la variable d_{ji} est due :

- à la contrainte elle-même. Nous ajoutons cette contrainte à l'explication calculée grâce à la méthode `theConstraint` (ligne 1 dans l'algorithme 34).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être calculées grâce à la méthode $theInf(d_{ij})$ (ligne 1 dans l'algorithme 34).

Pour la méthode `awakeOnSup`, la modification de la borne inférieure de la variable d_{ji} est due :

- à la contrainte elle-même. Nous ajoutons cette contrainte à l'explication calculée grâce à la méthode `theConstraint` (ligne 1 dans l'algorithme 35).
- aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être calculées grâce à la méthode $theSup(d_{ij})$ (ligne 1 dans l'algorithme 35).

11.4.3.2 La contrainte $i \leftrightarrow^{d_{max}}_{d_{min}} j$

Pour la méthode `awakeOnInf`, la modification des deux bornes inférieure et supérieure de la variable d_{ji} est due :

- à la contrainte elle-même. Cette contrainte est ajoutée à l'explication calculée grâce à la méthode `theConstraint` (ligne 1 dans l'algorithme 36).

Algorithme 35: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $i \leftrightarrow^{d} j$*, et v représente une de ses variables*

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
    si $d_{ij}^{max} < d$ **alors**
1     $updateInf(d_{ji},\ d,\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
      $updateSup(d_{ji},\ d,\ becauseOf(theConstraint(C),\ theSup(d_{ij})))$
  si $v = d_{ji}$ **alors**
    si $d_{ji}^{max} < d$ **alors**
      $updateInf(d_{ij},\ d,\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
      $updateSup(d_{ij},\ d,\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
fin

Algorithme 36: La borne inférieure de l'une des variables d_{ij} *ou* d_{ji} a augmenté.

C est une contrainte de type $i \leftrightarrow_{d_{min}}^{d_{max}} j$*, et v représente une de ses variables*

awakeOnInf(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
    si $d_{ij}^{min} > d_{max}$ **alors**
1     $updateInf(d_{ji},\ d_{min},\ becauseOf(theConstraint(C),\ theInf(d_{ij})))$
      $updateSup(d_{ji},\ d_{max},\ becauseOf(theConstraint(C),\ theInf(d_{ij})))$
  si $v = d_{ji}$ **alors**
    si $d_{ji}^{min} > d_{max}$ **alors**
      $updateInf(d_{ij},\ d_{min},\ becauseOf(theConstraint(C),\ theInf(d_{ji})))$
      $updateSup(d_{ij},\ d_{max},\ becauseOf(theConstraint(C),\ theInf(d_{ji})))$
fin

– aux contraintes responsables de la modification de la borne inférieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode $theInf(d_{ij})$ (ligne 1 dans l'algorithme 36).

Pour la méthode `awakeOnSup`, la modification des deux bornes inférieure et supérieure de la variable d_{ji} est due :

– à la contrainte elle-même. Cette contrainte est ajoutée à l'explication calculée grâce à la méthode `theConstraint` (ligne 1 dans l'algorithme 37).

– aux contraintes responsables de la modification de la borne supérieure de la variable d_{ij}. Ces contraintes peuvent être récupérées grâce à la méthode $theSup(d_{ij})$ (ligne 1 dans l'algorithme 37).

Algorithme 37: La borne supérieure de l'une des variables d_{ij} *ou* d_{ji} a diminué.

C est une contrainte de type $i \leftrightarrow_{d_{min}}^{d_{max}} j$, et v représente une de ses variables

awakeOnSup(C:**contrainte**, v:**distance**)
début
  si $v = d_{ij}$ **alors**
    si $d_{ij}^{max} < d_{min}$ **alors**
1     $updateInf(d_{ji},\ d_{min},\ becauseOf(theConstraint(C),\ theSup(d_{ij}))e)$
      $updateSup(d_{ji},\ d_{max},\ becauseOf(theConstraint(C),\ theSup(d_{ij}))e)$
  si $v = d_{ji}$ **alors**
    si $d_{ji}^{max} < d_{min}$ **alors**
      $updateInf(d_{ij},\ d_{min},\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
      $updateSup(d_{ij},\ d_{max},\ becauseOf(theConstraint(C),\ theSup(d_{ji})))$
fin

Exemple 48 (Contraintes disjonctives généralisées avec explication) :
Dans l'exemple 45 (page 162), la propagation de la contrainte c_5 : $(d_{36} = 2 \vee d_{63} = 2)$ réduit le domaine de la variable d_{36} à $\{2\}$. L'explication associée à la réduction de la borne inférieure de d_{36} peut être calculée comme suit :
- si la borne inférieure de d_{63} a augmentée, et est devenue strictement supérieure à 2 alors :

$$theInf(d_{63}) \wedge c_5$$

- si la borne supérieure de d_{63} a dinimuée, et est devenue strictement inférieure à 2 alors :

$$theSup(d_{63}) \wedge c_5$$

La propagation de la contrainte c_6 : $(2 \leq d_{34} \leq 3) \vee (2 \leq d_{43} \leq 3)$ met le domaine de d_{34} à $[2,3]$. L'explication associée à la réduction de la borne inférieure du domaine de d_{34} peut être calculée comme suit :
- si la borne inférieure de d_{43} a augmentée, et est devenue strictement supéreure à $d_{max} = 3$ alors :

$$theInf(d_{43}) \wedge c_6$$

- si la borne supérieure de d_{43} a dinimuée, et est devenue strictement inféreure à $d_{min} = 2$ alors :

$$theSup(d_{43}) \wedge c_6$$

11.5 Conclusion

Dans ce chapitre, nous avons présenté une généralisation des contraintes de précédence, de chevauchement et disjonctives basée sur le délai. Aussi, nous avons présenté les démarches nécessaires pour introduire ces contraintes dans notre système. Dans le chapitre suivant, nous proposons une généralisation des contraintes de ressources.

Chapitre 12

Contraintes de ressources généralisées

Sommaire

Introduction

Dans le chapitre 9, nous avons présenté deux types de contraintes de ressource : une contrainte *parties obligatoires* basée sur la notion de *parties obligatoires* [Klein et Scholl, 1999] et la notion d'*histogramme* [Caseau et Laburthe, 1994], et une contrainte *intervalles de tâches* basée sur la notion des *intervalles de tâches* [Caseau et Laburthe, 1994]. Ces contraintes ont été utilisées pour la résolution du RCPSP dans le cadre statique et dynamique. Elles peuvent être aussi utilisées pour la résolution des problèmes d'ordonnancement à contraintes de ressource (par exemple le RCPSP avec relations de précédence généralisées), où la capacité de chacune des ressources reste la même en chaque période de temps.

Or, dans certains problèmes réels, la capacité de chacune des ressources peut varier au cours du temps. La capacité maximale d'une ressource peut ne pas être constamment disponible. Les machines peuvent subir des périodes fixes de maintenance et les opérateurs ne travaillent pas 24 heures sur 24. Ces problèmes sont peu étudiées dans la littérature. Citons par exemple, le GRCPSP qui est une extension du RCPSP (*cf.* section 2.6 page 21).

Dans ce chapitre, nous présentons une extension des contraintes de ressource *parties obligatoires* et *intervalles de tâches* prenant compte des variations de la capacité des ressources.

12.1 Représentation d'une ressource à capacité variable

Une ressource peut être considérée comme un calendrier dans lequel : à chaque période de temps $\prec t \succ$ est associée une capacité $R_k[t]$ représentant la disponibilité de la ressource k à cette période de temps. Cette capacité vérifie la relation $R_k[t] \leq R_k$, où R_k est la capacité maximale de la ressource.

Ainsi, nous représentons une ressource k par une liste $\mathcal{R}_k$ représentant la capacité de la ressource pour chaque période de temps et l'ensemble des activités utilisant cette ressource.

Exemple 49 (Représentation d'une ressource) :
La figure 12.1 représente une ressource k à capacité variable.
La disponibilité de cette ressource est représentée par la liste :
$\mathcal{R}_k = (4,5,5,4,4,5,5,3,3,4,4,4,5,5,5,5,3,3)$.
La capacité maximale est $R_k = 5$.

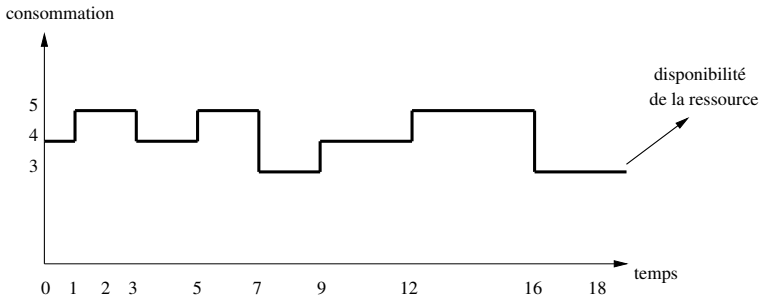


FIG. 12.1 – *Représentation d'une ressource à capacité non constante.*

Dans la suite de ce chapitre, R_k désigne la capacité maximale de la ressource k :

$$R_k = \max_{t \leq h}\{R_k[t]\},\ \textit{où } h \textit{ est l'horizon du projet.}$$

12.2 Un exemple de GRCPSP

Dans cette section, nous considérons un exemple de GRCPSP constitué de huit activités utilisant une seule ressource à capacité variable. Nous associons à chaque activité i, sa durée opératoire p_i et sa quantité de ressource a_{i1}. Le tableau 12.1 fournit les données du problème. La capacité de la ressource est resprésentée par la liste :
$\mathcal{R}_1 = (4,5,5,4,4,5,5,3,3,4,4,4,5,5,5,5,3,3)$.

i	p_i	a_{i1}	i	p_i	a_{i1}
1	0	0	5	8	2
2	2	3	6	4	4
3	4	2	7	4	2
4	2	1	8	0	0

TAB. 12.1 – *Données de l'exemple 12.2*

Les activités sont liées par des relations de précédence représentées dans le graphe 12.2.

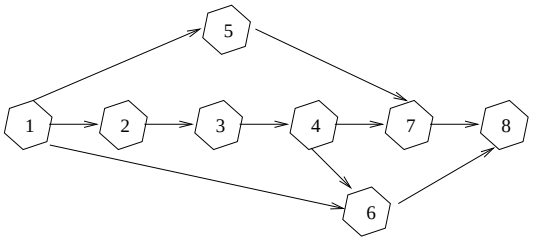


FIG. 12.2 – *Graphe de précédence*

La figure 12.3 représente une solution réalisable du problème.

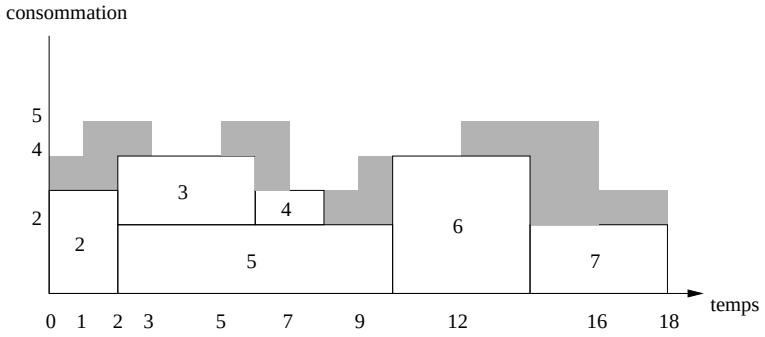


FIG. 12.3 – *Une solution réalisable du problème.*

12.3 Règle déduite de la capacité de la ressource

Cette règle de déduction (*cf.* section 7.3 page 76) peut être étendue au cas où la capacité de la ressource n'est pas constante.

Ainsi, si pour une ressource k, la somme des besoins en ressource $a_{ik}+a_{jk}$ des deux activités i et j dépasse la capacité maximale R_k alors nous sommes sûrs que ces deux activités sont en disjonction $(i \leftrightarrow j)$. Ceci consiste à ajouter la contrainte de disjonction $d_{ij} \times d_{ji} \leq 0$ au réseau de contraintes, où d_{ij} et d_{ji} représentent des distances (*cf.* section 7.2 page 71).

12.4 Contrainte basée sur les parties obligatoires

Dans la section 9.2 (page 91), nous avons défini la contrainte *parties obligatoires* $\mathcal{C}_{po}(k)$, sur une ressource k, comme une contrainte globale. Les variables associées à cette contrainte sont toutes les variables t_i associées aux activités i qui utilisent la même ressource k. Cette contrainte peut être étendue à une contrainte globale que nous noterons $\mathcal{C}^{\star}_{po}(k)$.

12.4.1 Principe

Comme pour la contrainte $\mathcal{C}_{po}(k)$, la contrainte étendue $\mathcal{C}^{\star}_{po}(k)$ sera définie par les deux histogrammes *histogramme des niveaux* et *histogramme des activités*.

Rappelons que l'*histogramme des niveaux* représente le niveau de la ressource réservé pour chaque période de temps $\prec t \succ$, et l'*historgamme des activités* repésente, pour chaque période de temps $\prec t \succ$, l'ensemble des activités qui ont réservé une quantité de ressource dans cette même période. Ces deux histogrammes sont représentés respectivement par les deux listes $\mathcal{H}n_k$ et $\mathcal{H}a_k$.

L'histogramme des niveaux a pour but de vérifier, après chaque modification de l'une des bornes du domaine d'une des activités, que le niveau de consommation ne dépasse pas la quantité de ressource disponible :

s'il existe une période de temps $\prec t \succ$ *telque* $\mathcal{H}n_k[t] > \mathcal{R}_k[t]$ *alors nous avons un conflit*

Cet histogramme est aussi utilisé pour ajuster les fenêtres temporelles des activités :

s'il existe une période de temps $\prec t \succ$ *incluse dans la fenêtre temporelle d'une activité* i *telque* $\mathcal{H}n_k[t] + a_{ik} > \mathcal{R}_k[t]$ *alors une des bornes du domaine de la variable* t_i *est ajustée.*

Signalons que le principe de maintien des structures des données que nous avons présenté pour la contrainte *parties obligatoires* (*cf.* section 9.2.3 page 92) reste le même pour la contrainte étendue. Par contre, des modifications sont nécessaires dans l'algorithme de propagation. Dans l'algorithme 38, nous donnons le mécanisme de propagation utilisé dans le cas où la date de début au plus tôt d'une activité a augmenté sachant que sa partie obligatoire n'était pas vide avant cette augmentation (*cf.* section 9.2.4 page 95).

12.4.2 Calcul des explications

Le calcul des explications associées aux conflits de ressources et aux ajustements des fenêtres temporelles reste le même pour la contrainte étendue.

Algorithme 38: Règles de propagation.

$\mathcal{C}^{\star}_{po}(k)$ *est une contrainte parties obligatoires étendue.*
i est l'activité dont la borne inférieure a augmentée de r_i à $r_i^{'}$.

Règles_Propagation($\mathcal{C}^{\star}_{po}(k)$:**contrainte**, i:**Activité**, r_i:**entier**, $r_i^{'}$:**entier**)

```
début
    pour chaque t ∈ [r_i + p_i + 1 .. r'_i + p_i] faire
        si Hn_k[t] > R_k[t] alors
            contradiction
    si Pas de contradiction alors
1       pour chaque t ∈ [r'_i + p_i + 1, f_i + p_i] faire
            si Hn_k[t] + a_ik > R_k[t] alors
                ajuster f_i à t − p_i − 1
fin
```

L'explication associée au conflit de ressource à une période de temps $\prec t \succ$ est :

$$\left(\bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j) \right) \bigwedge \mathcal{C}^{\star}_{po}(k)$$

Si nous nous situons dans le cas où l'ajustement de la fenêtre temporelle d'une activité i a été causé par l'augmentation de la borne inférieure de son domaine, et par l'augmentation de la quantité de ressource utilisée sur une période de temps $\prec t \succ$ dans l'histogramme des niveaux, alors l'explication associée à cet ajustement est :

$$\left(\bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j) \right) \bigwedge \texttt{theInf}(t_i) \bigwedge \mathcal{C}^{\star}_{po}(k)$$

L'algorithme 39 représente la version expliquée de l'algorithme 38.

12.5 Contrainte basée sur la notion d'intervalle de tâches

Dans la section 9.3 (page 99), nous avons modélisé la contrainte *intervalles de tâches* comme une contrainte globale. Nous avons associé à chaque contrainte $\mathcal{C}_{it}[IT, k]$ l'ensemble de toutes les variables associées aux activités utilisant la ressource k. Cette contrainte sera étendue à une contrainte, notée $\mathcal{C}^{\star}_{it}[IT, k]$, tenant compte de la variation de la capacité de la ressource.

12.5.1 Principe

Comme pour la contrainte $\mathcal{C}_{it}[IT, k]$, la contrainte $\mathcal{C}^{\star}_{it}[IT, k]$ sera définie par :

- *début*($\mathcal{C}^{\star}_{it}[IT, k]$) et $fin(\mathcal{C}^{\star}_{it}[IT, k])$ qui correspodent respectivement aux deux activités i et j de l'intervalle de tâches $IT = [i, j]$;

Algorithme 39: Règles de propagation avec explications.

$\mathcal{C}^{\star}_{po}(k)$ *est une contrainte parties obligatoires étendue.*
i est l'activité dont la borne inférieure a augmentée de r_i *à* $r_i^{'}$.

Règles_Propagation($\mathcal{C}^{\star}_{po}(k)$:**contrainte**, i:**Activité**, r_i:**entier**, $r_i^{'}$:**entier**)
début
  pour chaque $t \in [r_i + p_i + 1..r_i^{'} + p_i]$ **faire**
    si $\mathcal{H}n_k[t] > \mathcal{R}_k[t]$ **alors**
      Contradiction, l'explication associée est : $\bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j) \bigwedge \mathcal{C}^{\star}_{po}(k)$
  si *Pas de contradiction* **alors**
    pour chaque $t \in [r_i^{'} + p_i + 1, f_i + p_i]$ **faire**
      si $\mathcal{H}n_k[t] + a_{ik} > \mathcal{R}_k[t]$ **alors**
        $updateSup(t_i,\ t - p_i - 1, becauseOf(theConstraint(\mathcal{C}^{\star}_{po}(k)), \bigwedge_{j \in \mathcal{H}a_k[t]} \texttt{theDom}(t_j) \bigwedge \texttt{theInf}(t_i))$
fin

- $intérieur(\mathcal{C}^{\star}_{it}[IT, k])$ qui constitue l'ensemble des activités qui sont à l'intérieur de l'intervalle de tâches IT (y compris les deux activités i et j);

- $extérieur(\mathcal{C}^{\star}_{it}[IT, k])$ qui contient l'ensemble des activités qui sont à l'extérieur de l'intervalle de tâches IT. En d'autre terme, ce sont les activités qui n'appartienentt pas à $intérieur(\mathcal{C}^{\star}_{it}[IT, k])$;

- $énergie(\mathcal{C}^{\star}_{it}[IT, k])$ représentant la quantité d'énergie consommée à l'intérieur de l'intervalle de tâches IT. Elle est égale à la somme des énergies $w_{lk} = p_l \times a_{lk}$ de toutes les activités appartenant à $intérieur(\mathcal{C}^{\star}_{it}[IT, k])$. Ainsi, nous avons :

$$énergie(\mathcal{C}^{\star}_{it}[IT, k]) = \sum_{l \in intérieur(\mathcal{C}^{\star}_{it}[IT,k])} w_{lk}.$$

Signalons que le mécanisme de maintien des structures de données pour la contrainte *intervalles de tâches étendue* reste le même que celui présenté pour la contrainte *intervalles de tâches* (*cf.* section 9.3.3 page 101).

Les règles de propagation que nous avons utilisées dans la contrainte $\mathcal{C}_{it}[IT, k]$ sont : la *règle d'intégrité*, la *règle d'intégrité améliorée* et la *règle du jet* (*cf.* section 9.3.4 page 103). Nous avons étendu ces règles pour le cas où la capacité de la ressource est non constante.

12.5.2 Règle d'integrité étendue

Cette règle détecte un conflit de ressource lorsque l'énergie associée à la contrainte $\mathcal{C}^{\star}_{it}[IT, k]$ dépasse la quantité d'énergie disponible sur l'intervalle de tâches IT. Cette énergie est égale à : $énergie(IT, k) = \sum_{r_i < t \leq d_i} R_k[t]$.

$$Si\ énergie(\mathcal{C}^{\star}_{it}[IT, k])\ >\ énergie(IT, k)\ alors\ nous\ avons\ un\ conflit$$

12.5.3 Règle de jet étendue

Comme pour la règle de jet, cette règle consiste à caler à gauche (ou à droite) une activité l qui intersecte un intervalle de tâches IT, puis de comparer la quantité d'énergie $énergie\_gauche(IT,k,l)$ (ou $énergie\_droite(IT,k,l)$) intersectant IT à la quantité d'énergie restante :

$$reste(IT,k) \ = \ énergie(\mathcal{C}^{\star}_{it}[IT,k]) \ - \ énergie\_gauche(IT,k,l).$$

Dans le cas où l'activité l est calée à gauche, nous pouvons calculer la quantité d'énergie $énergie\_gauche(IT,k,l)$ intersectant l'intervalle de tâches IT grâce à la formule :

$$énergie\_gauche(IT,k,l) = min\{w_{lk} - (r_i - r_l) \times a_{lk}, (f_j + p_j - r_l) \times a_{lk}, w_{lk}\}$$

Nous avons donc l'ajustement :

$$Si\ reste(IT,k) < énergie\_gauche(IT,k,l)\ alors\ r_l \ := \ max\{r_l, f_j + p_j - \lfloor \frac{reste(IT,k)}{a_{lk}} \rfloor\}$$

Symétriquement, lorsque l'activité l est calée à droite, la quantité $énergie\_droite(IT,k,l)$ intersectant l'intervalle de tâches IT peut être calculé par la formule :

$$énergie\_droite(IT,k,l) = min\{w_{lk} - (f_l + p_l - f_j - p_j) \times a_{lk}, (f_l + p_l - r_i) \times a_{lk}, w_{lk}\}$$

Et nous avons l'ajustement suivant :

$$Si\ reste(IT,k) < énergie\_droite(IT,k,l)\ alors\ f_l \ := \ min\{f_l, r_i - p_l + \lfloor \frac{reste(IT,k)}{a_{lk}} \rfloor\}$$

12.5.4 Règle d'integrité étendue améliorée

Comme pour la règle d'integrité améliorée, cette règle tient compte des quantités d'énergies minimales que doivent réserver les activités l de $extérieur(\mathcal{C}^{\star}_{it}[IT,k])$. Cette quantité peut être calculé par la formule (*cf.* section 9.3.4.3 page 107) :

$$énergie(IT,k,l) \ = \ a_{lk} \times \max\{0, \min\{d_j - r_i, r_l + p_l - r_i, f_j + p_j - f_l\}\}$$

Si nous notons par $intersecte(\mathcal{C}^{\star}_{it}[IT,k])$ l'ensemble des activités de $extérieur(\mathcal{C}^{\star}_{it}[IT,k])$ qui intersectent l'intervalle de tâches IT, la règle d'integrité étendue améliorée sera donc :

$$Si\ énergie(IT,k) \ + \sum_{l \in intersecte(\mathcal{C}^{\star}_{it}[IT,k])} énergie(IT,k,l) \ > \ \sum_{r_i < t \leq d_j} \mathcal{R}_k[t]$$

alors nous avons un conflit de ressource

12.5.5 Règle de jet étendue améliorée

Lorsque nous comptabilisons les énergies minimales intersectant l'intervalle de tâches, la quantité d'énergie restante $reste(IT,k)$ devient :

$$reste'(IT,k,l) = \sum_{r_i < t \leq d_j} \mathcal{R}_k[t] - énergie(IT,k) \ - \sum_{l' \in intesecte(\mathcal{C}_{it}[IT,k]) \setminus \{l\}} énergie(IT,k,l')$$

Algorithme 40: Règles de propagation.

l est l'activité dont la borne inférieure augmente de r_l à r_l'.
$\mathcal{C}_{it}^{\star}[IT,k]$ est une contrainte intervalles de tâches tels que $l \in extérieur(\mathcal{C}_{it}^{\star}[IT,k])$.

Règle_Propagation($\mathcal{C}_{it}^{\star}[IT,k]$:**contrainte**, l:**Activité**, r_l:**entier**, r_l':**entier**)
début
1 **si** $reste(IT,k) < 0$ **alors**
  contradiction
sinon
2 **si** $énergie(IT,k) + \sum_{l \in intersecte(\mathcal{C}_{it}^{\star}[IT,k])} énergie(IT,k,l) > \sum_{r_i < t \leq d_j} \mathcal{R}_k[t]$ **alors**
  contradiction
sinon
3 **si** $reste'(IT,k) < énergie\_gauche(IT,k,l)$ **alors**
  ajuster r_l *à* $(d_j - \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor)$
fin

Dans le cas où l'activité l est calée à gauche, nous avons l'ajustement :

$$Si\ reste'(IT,k) < énergie\_gauche(IT,k,l)\ alors\ r_l\ :=\ max\{r_l, f_j + p_j - \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor\}$$

Et dans le cas où l'activité l est calée à droite, nous avons l'ajustement :

$$Si\ reste'(IT,k) < énergie\_droite(IT,k,l)\ alors\ f_l\ :=\ min\{f_l, r_i - p_l + \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor\}$$

L'algorithme 40 décrit le mécanisme de propagation dans le cas de l'augmentation de la borne inférieure du domaine d'une activité.

12.5.6 Calcul des explications

Le calcul des explications associées aux règles de déduction précedentes suivent le même principe que pour le cas non étendu. Nous présentons juste les formules permettant de les calculer.

• La règle d'intégrité détecte un conflit de ressource lorsque la quantité d'énergie nécessaire pour exécuter toutes les activités de $intérieur(\mathcal{C}_{it}^{\star}[IT,k])$ dépasse l'énergie disponible sur l'intervalle de tâches IT. L'explication associée à ce conflit est :

$$\left(\bigwedge_{j \in intérieur(\mathcal{C}_{it}^{\star}[IT,k])} \texttt{theDom}(t_j) \right) \bigwedge \mathcal{C}_{it}[IT,k]$$

• La règle d'intégrité étendue améliorée prend en compte les activités de $extérieur(\mathcal{C}_{it}^{\star}[IT,k])$ qui intersectent l'intervalle de tâches IT. L'explication associée à ce conflit est :

$$\bigwedge_{j \in intérieur(\mathcal{C}_{it}^{\star}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}_{it}^{\star}[IT,k])} \texttt{theDom}(t_j) \bigwedge \mathcal{C}_{it}^{\star}[IT,k]$$

Algorithme 41: Règles de propagation avec explications.

l est l'activité dont la borne inférieure a augmenté de r_l à r_l'.
$\mathcal{C}^\star_{it}[IT,k]$ est une contrainte intervalles de tâches telque $l \in extérieur(\mathcal{C}^\star_{it}[IT,k])$.

Règle_Propagation($\mathcal{C}^\star_{it}[IT,k]$:**contrainte**, l:**Activité**, r_l:**entier**, r_l':**entier**)
début
  si $reste(IT,k) < 0$ **alors**
    Contradiction, l'explication de ce conflit est :

$$\bigwedge_{j \in intérieur(\mathcal{C}^\star_{it}[IT,k])} \texttt{theDom}(t_j) \bigwedge \mathcal{C}^\star_{it}[IT,k]$$

  sinon
    si $énergie(IT,k) + \sum_{l \in extérieur(\mathcal{C}^\star_{it}[IT,k])} énergie(IT,k,l) > \sum_{r_i < t \leq d_j} \mathcal{R}_k[t]$ **alors**
      Contradiction, l'explication de ce conflit :

$$\bigwedge_{j \in intérieur(\mathcal{C}^\star_{it}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}^\star_{it}[IT,k])} \texttt{theDom}(t_j) \bigwedge \mathcal{C}^\star_{it}[IT,k]$$

    sinon
      si $reste'(IT,k) < énergie\_gauche(IT,k,l)$ **alors**
        $updateInf(t_l, d_j - \lfloor \frac{reste'(IT,k)}{a_{lk}} \rfloor, becauseOf(thConstraint(\mathcal{C}^\star_{it}[IT,k]),$

$$\bigwedge_{j \in intérieur(\mathcal{C}^\star_{it}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}^\star_{it}[IT,k]) \setminus \{i\}} \texttt{theDom}(t_j) \bigwedge \texttt{theInf}(t_l))$$

fin

• La règle du jet étendue améliorée permettant d'ajuster les bornes des domaines des activités. Lorsque la date de début au plus tôt r_l d'une activité l a été ajustée, l'explication associée est :

$$\bigwedge_{j \in intérieur(\mathcal{C}^\star_{it}[IT,k])} \texttt{theDom}(t_j) \quad \bigwedge_{j \in intersecte(\mathcal{C}^\star_{it}[IT,k]) \setminus \{i\}} \texttt{theDom}(t_j) \bigwedge \texttt{theInf}(t_l) \bigwedge \mathcal{C}^\star_{it}[IT,k]$$

La version expliquée de l'algorithme 40, décrivant les étapes suivit pour propager les contraintes *intervalles de tâches*, est l'algorithme 41.

12.6 Conclusion

Dans ce chapitre, nous avons étendu les règles de déduction que nous avons utilisées pour le RCPSP, afin de les appliquer au cas où la disponibilité des ressources peut varier. Nous avons vu que ces extensions sont très simples à mettre en œuvre car tous les algorithmes dont nous avons besoin ont été déjà implémentés pour le RCPSP.

Chapitre 13

Expérimentations

Sommaire

Introduction

Dans ce chapitre, nous présentons quelques expérimentations sur le RCPSP avec des contraintes temporelles généralisées. Nous présentons aussi les techniques que nous pouvons utiliser dans le cas où la capacité de la ressource varie sur un intervalle de temps.

13.1 Événements pris en compte

Cette section présente les différents événements pouvant être pris en compte dans notre système pour la résolution de quelques extensions du RCPSP dans le cadre dynamique. Nous avons classé ces événements en trois types : des événements temporels, des événements se rapportant aux activités et des événements se rapportant aux ressources.

13.1.1 Événements temporels

Ce type d'événements regroupe tout les événements qui peuvent être modélisés par l'ajout, le retrait ou la modification des contraintes temporelles généralisées telles que celles définies dans la section 11.1 (page 152). Nous considérons aussi les événements qui obligent une activité de commencer à une date bien précise ou à une des dates d'un intervalle.

Dans le tableau 13.1, nous listons les contraintes associées à chacun de ces événements temporels.

Événement	Contrainte
i doit commencer à la date v_i	$t_i = v_i$
i doit commencer entre a_i *et* b_i	$a_i \leq t_i \leq b_i$
$i \rightarrow^{d} j$	$d_{ij} = d$
$i \rightarrow_{d_{min}}^{d_{max}} j$	$d_{min} \leq d_{ij} \leq d_{max}$
$i \leftrightarrow^{d} j$	$d_{ij} = d \vee d_{ji} = d$
$i \leftrightarrow_{d_{min}}^{d_{max}} j$	$(d_{min} \leq d_{ij} \leq d_{max}) \vee (d_{min} \leq d_{ji} \leq d_{max})$
$i \parallel^{d} j$	$d_{ij} = -d$
$i \parallel_{d_{min}}^{d_{max}} j$	$-d_{max} \leq d_{ij} \leq -d_{min}$

TAB. 13.1 – *Contraintes correspondant aux événements temporels*

13.1.2 Événements se rapportant aux activités

Nous pouvons facilement ajouter, retirer ou modifier une activité. La procédure suivie est la même que celle présentée dans la section 10.4.1.2 (page 130) pour le RCPSP classique.

13.1.3 Événements se rapportant aux ressources

Comme nous l'avons vu dans la section 10.4.1.3 (page 131) nous pouvons ajouter et retirer une ressource. Nous suivons les mêmes procédures que celles utilisées pour le RCPSP classique.

Par contre, lorsque la capacité de la ressource subit des variations la procédure devient plus délicate. Nous pouvons éventuellement retirer l'ancienne ressource et ajouter une nouvelle ressource avec les nouvelles données.

Dans le cas où la capacité de la ressource reste constante pendant toute la durée du projet, nous pouvons déclarer cette capacité comme une variable. Nous ajoutons la contrainte $x_k = R_k$ au réseau de contraintes. Lorsque la capacité de la ressource k passe de R_k à $R_k^{'}$, nous retirons cette contrainte et ajoutons la contrainte $x_k = R_k^{'}$. Nous signalons que la variable x_k doit être introduite dans les contraintes de ressource.

Dans le cas où la capacité de la ressource n'est plus constante durant l'horizon du projet, nous pouvons déclarer, pour chaque période de temps $\prec t \succ$, la quantité de ressource associée $R_k[t]$ comme une variable. Cette approche n'est valable que si l'horizon du projet n'est pas assez grand.

Une solution judicieuse sera de gérer la diminution et l'augmentation de la capacité d'une ressource par l'ajout et le retrait d'activités. En effet :

– dans le cas où la capacité de la ressource k reste constante durant toute la durée du projet, nous pouvons considérer une activité fictive de durée opératoire égale à la valeur du C_{max}, et de quantité de ressource égale à Δ_k où Δ_k est la quantité de ressource tolérée (*Fig.* 13.1).

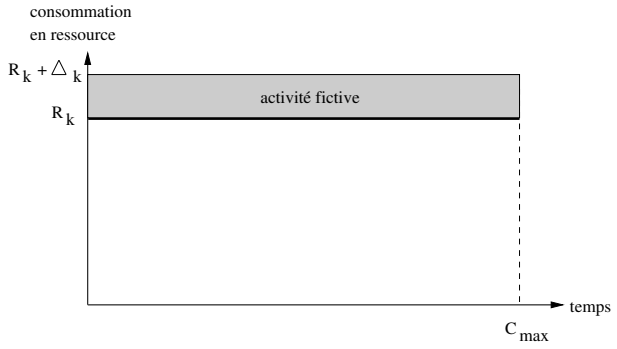


FIG. 13.1 – *Variation de la capacité de la ressource pour une ressource à capacité constante*

– dans le cas où la capacité de la ressource n'est pas constante mais varie en fonction du temps, nous pouvons considérer une capacité maximale et ajouter des activités fictives qui couvrent les surfaces non disponibles de la ressource (*Fig.* 13.2).

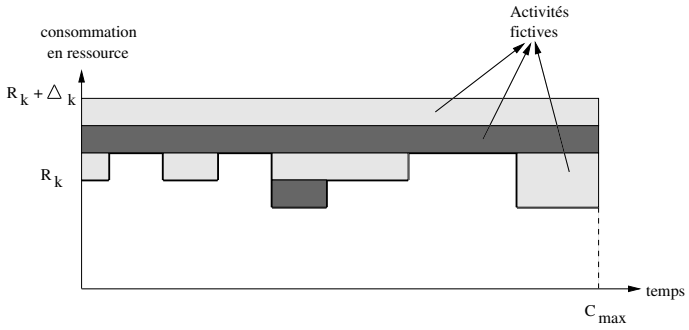


FIG. 13.2 – *Variation de la capacité de la ressource pour une ressource à capacité variable*

Lorsque la capacité d'une ressource a diminué sur une période de temps, nous ajoutons une activité fictive ayant pour durée opératoire la longueur de la période de temps, pour quantité de ressource la valeur de la variation de la ressource, et pour date de début la date de début de la période de temps. Par contre, lorsque la capacité d'une ressource a augmenté sur une période de temps, nous retirons certaines activités et ajoutons d'autres.

13.2 Résultats expérimentaux

Nous n'avons considéré dans nos expérimentations que les événements temporels, et plus particuliérement les événements qui correspondent à l'ajout de contraintes temporels. Pour la variation de la capacité des ressources, ils sont traités comme des ajouts et des retraits d'activités.

13.2.1 Protocole expérimental

Chaque série de problèmes est construite de la façon suivante :

- le problème initial P_0 est un problème de la série SMCP;
- le problème P_i est construit à partir du problème P_{i-1} en lui ajoutant aléatoirement une des contraints temporelles présentées dans le tableau 13.1.

Nous signalons que nous n'avons gardé que les séries dont les solutions produites (par les deux approches) sont différentes à chaque étape.

Pour les problèmes comportant 12 activités, les séries sont construites en ajoutant successivement 4 contraintes temporelles. Pour les problèmes comportant 22 ou 32 activités, nous avons ajouté successivement 5 contraintes temporelles.

Nous avons construit 15 séries de problèmes comportant chacun 12 activités, 12 séries de problèmes qui comportent chacun 22 activités, et 15 séries de problèmes comportant 32 activités.

Nous avons repris les mêmes paramètres G_{temps} et G_{mes} définies dans la section 10.4.4 (page 135), et nous avons considéré les mêmes mesures définies dans la section 10.4.2 (page 132)

13.2.2 Analyse des résultats

Nous comparons notre approche à l'approche classique qui consiste à recalculer une nouvelle solution à chaque perturbation. Nous signalons que tous les résultats présentés dans cette section sont détaillés dans l'annexe B.

Analyse des résultats en terme de temps CPU Dans le tableau 13.2, nous présentons le gain minimal Min, le gain maximal Max, la moyenne des gains Moy, et le gain entre les moyennes des temps $GMoy$ (*cf.* section 10.4.4 page 135).

Nous constatons, dans ce tableau, que nous avons des gains en temps important. La moyenne des gains se situe entre 0.4% et 26.3%. Aussi, nous pouvons voir que le gain $GMoy$ est important. Il est plus important pour les séries de problèmes de taille 12 et 22, et moins important pour les séries de taille 32.

	$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		$P_4 \rightarrow P_5$		
	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	Min/Max	Moy	GMoy
12 act.	-12.1%	23.6%	-22.3%	25.8%	-29.2%	23%	-21.7%	21.3%	–	–	30.6%
	40.5%		76.6%		69%		62.7%		–		
22 act.	-2.3%	18.3%	0.6%	21.5%	2.5%	26.3%	4.1%	22.7%	5.6%	25.3%	27.5%
	51.1%		56.3%		62.6%		58..9%		63.4%		
32 act.	-61.3%	0.4%	-41.2%	1.2%	-26.3%	11.1%	-35.1%	13.7%	-21.1%	19.2%	9.2%
	42.8%		56.8%		47.5%		56.8%		57.7%		

TAB. 13.2 – *Résultats pour le gain en temps pour des problèmes dans lesquels une série de contraintes temporelles généralisées est ajoutée.*

Étude de la stabilité pour la mesure DIFF Dans le tableau 13.3, nous résumons les résultats obtenus pour résolution des séries de problèmes dans lesquelles une contrainte temporelle est ajouée à chaque étape. Ces expérimentations montrent un gain assez important pour les séries de problèmes de taille 32. Nous pouvons aussi constater que les gains entre les solutions initiales et les solutions finales sont très faibles pour les trois types de séries.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_0 \rightarrow P_{fin}$
12	-3.7%	-2.7%	13.4%	0%	–	0.8%
22	-14.2%	34%	-9%	-10.3%	-6.5%	-2.6%
32	8%	10.7%	7.1%	10.9%	-7.4%	2.1%

TAB. 13.3 – *Résultats pour le gain en performance pour la mesure* DIFF *sur des problèmes dans lesquels une série de contraintes temporelles est ajoutée.*

Étude de la stabilité pour la mesure POSI Le tableau 13.4 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte temporelle est ajoutée à chaque étape. Nous constatons que nous avons des gains important pour tous les types de séries. Ces gains sont situés entre -14.6% et 52%. Nous avons des gains très important pour les séries de problèmes de taille 32. Aussi, nous avons des gains entre les solutions initiales et finales.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_0 \rightarrow P_{fin}$
12	-9%	3.1%	2.3%	8.8%	- -	7%
22	-9.1%	38.2%	9.5%	-14.6%	5.6%	8.9%
32	11.3%	52%	12.1%	40.5%	23%	7.1%

TAB. 13.4 – *Résultats pour le gain en performance pour la mesure* POSI *sur des problèmes dans lesquels une série de contraintes temporelles est ajoutée.*

Étude de la stabilité pour la mesure SDIF Dans le tableau 13.5, nous présentons les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte temporelle est ajoutée à chaque étape. Nous pouvons constater des gains important pour tous les types de séries. Ces gains sont situés entre -10.9% et 42.6%. Nous cpnstatons aussi des gains très important pour les séries de problèmes de taille 32. Nous avons aussi des gains importants entre les solutions initiales et finales.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_0 \rightarrow P_{fin}$
12	-9.3%	2.6%	12.8%	7.1%	–	0.4%
22	-10.9%	32.6%	1.6%	-2%	-9.7%	11.4%
32	10.9%	42%	7.5%	42.6%	18.2%	10.6%

TAB. 13.5 – *Résultats pour le gain en performance pour la mesure* SDIF *sur des problèmes dans lesquels une série de contraintes temporelles est ajoutée.*

Étude de la stabilité pour la mesure MDIF Le tableau 13.6 résume les résultats obtenus pour la résolution des séries de problèmes dans lesquelles une contrainte temporelle est ajoutée à chaque étape. Nous constatons des gains important pour tous les types de séries. Ces gains sont situés entre -41.4% et 46.1%. Aussi, nous avons des gains importants entre les solutions initiales et finales.

	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_0 \rightarrow P_{fin}$
12	-0.5%	13.7%	12.5%	-10.9%	–	0.6%
22	-5.4%	46.1%	13.2%	9.8%	-41.4%	9.6%
32	17.5%	35.5%	-16%	10.1%	34.7%	2.5%

TAB. 13.6 – *Résultats pour le gain en performance pour la mesure* MDIF *sur des problèmes dans lesquels une série de contraintes temporelles est ajoutée.*

13.3 Conclusion

Dans ce chapitre, nous avons mené des expérimentations sur des séries de problèmes dans lesquelles un contrainte temporelle est ajoutée à chaque étape. Les résultats que nous avons obtenus sont moins importants que ceux obtenus pour le RCPSP classique. Cela peut s'expliquer par le fait que les contraintes temporelles généralisées sont plus contraignantes que les contraintes de précédences. Notre système met donc plus de temps pour retrouver une nouvelle solution, et par conséquent perturbe plus la solution courante.

Quatrième partie

Conclusion

Chapitre 14

Conclusion et perspectives

Dans cette thèse, nous nous sommes intéressés à la résolution des problèmes d'ordonnancement dynamiques. Et plus particulièrement à la résolution des problèmes d'ordonnancement de projet à contraintes de ressource (RCPSP) et ses extensions dans un cadre dynamique.

À cet effet, nous avons conçu un système dédié à la résolution des problèmes d'ordonnancement dynamiques. Notre système combine des techniques issues de la recherche opérationnelle et de la programmation par contraintes avec explications.

Notre travail comprend les contributions suivantes :

1. Un **schéma de séparation** [Elkhyari *et al.*, 2002c, Elkhyari *et al.*, 2003a] adaptable à la stratégie de recherche *mac-dbt* et permettant la résolution du RCPSP et ses extensions. Ce schéma de séparation est basé sur les positions relatives entre les couples d'activités.

2. Une **notion de distance** [Elkhyari *et al.*, 2002a, Elkhyari *et al.*, 2002b] qui nous a permis dans un premier temps de modéliser des contraintes temporelles (précédence, chevauchement et disjonction) en des contraintes simples. Nous avons développé des algorithmes permettant la propagation de ces contraintes et le calcul des explications associées. Ces contraintes sont utilisées pour modéliser les contraintes temporelles du problème, les contraintes de décision prises au cours de la recherche, et certains aléas.

3. Deux **contraintes globales** [Elkhyari *et al.*, 2003a] permettant de maintenir les contraintes de ressources. Ces contraintes sont essentiellement basées sur des règles de déduction de la litérature. Leurs buts est de détecter des conflits de ressources et d'ajuster les fenêtres temporelles des activités. Nous avons présenté une **amélioration** de ces règles dans le but de détecter plus de conflits et d'affiner les ajustements. Nous avons conçu un algorithme, pour chacune de ces contraintes, qui permet de détecter les conflits, d'ajuster des fenêtres temporelles et de calculer les explications.

4. Une **généralisation des contraintes temporelles** [Elkhyari *et al.*, 2003c] (précédence, chevauchement et disjonction) basée sur la notion de délai. Grâce à la notion de distance, ces contraintes ont été modélisées en contraintes simples. Nous avons présenté des algorithmes permettant leur propagation et le calcul des explications associées.

5. Une **généralisation des contraintes de ressources** [Elkhyari *et al.*, 2003c] qui permet de résoudre des problèmes d'ordonnancement plus complexes. Ce sont des problèmes d'ordonnancement à contraintes de ressources où la capacité des ressources est variable.

Il s'agit des extensions des contraintes globales développées précédemment. Nous avons donc étendu les règles de déduction utilisées dans ces contraintes globales pour prendre en compte la variation des capacités des ressources.

6. Et enfin, l'ensemble de toutes ces contributions nous a permis de développer un **système** capable de résoudre le RCPSP et ses extensions dans un cadre **dynamique** (*Elkhyari et al.* [2003a, 2003b]). Notre système est aussi utilisable pour la résolution des **problèmes sans solution**.

Nous avons mené plusieurs expérimentations sur des séries de jeux de tests de la littérature pour le RCPSP. Les résultats obtenus pour la résolution du RCPSP dans le cadre statique montrent que notre approche a un comportement raisonnable par rapport aux approches existantes. Nous avons aussi, dans le cadre dynamique, mené des tests sur des séries de problèmes construites à partir de jeux de tests existants. Nous avons comparé notre approche à l'approche classique (ré-ordonnancement) en terme de temps de calcul et en terme de stabilité. Nous avons introduit des mesures de performance permettant de mesurer la stabilité des solutions successives.

Nous avons effectué deux séries de tests :

- la première concerne le RCPSP. Nous nous sommes intéressés aux perturbations suivantes : l'ajout et le retrait d'une série d'activités, et l'ajout et le retrait d'une série de contraintes de précédence. Les résultats obtenus sont très satisfaisants. En terme de temps de calcul, nous avons obtenu des gains importants sur les séries de problèmes de grande taille et de taille moyenne. En terme de stabilité, les résultats obtenus montrent des gains en performance très importants sur des séries de problèmes de grande taille pour les quatre mesures considérées, des gains moins importants sur des séries de problèmes de taille moyenne pour quelques mesures, et quelques pertes sur des séries de petite taille.

- la deuxième série concerne les extensions du RCPSP. Les perturbations auxquelles nous nous sommes intéressés sont : l'ajout de contraintes temporelles généralisées.

Nous pensons qu'il existe de nombreuses pistes à la suite de notre travail. Nous pouvons citer :

- Améliorer les contraintes de ressources en ajoutant des règles de déduction plus efficaces et en proposant d'autres algorithmes de propagation plus sophistiqués.

- Assurer une interaction entre le système et l'utilisateur. Le système pourait alors aider l'utilisateur en lui fournissant des informations sur le déroulement de la recherche au cours de la résolution, sur les contraintes qui ont conduit aux échecs au cours de la résolution ...

- Appliquer notre approche à d'autres problèmes d'ordonnancement plus complexes, par exemple : à des problèmes d'ordonnancement avec des contraintes de type réservoirs, à des problèmes où les durées opératoires et les besoins en ressources des activités sont des variables ...

Nous pouvons aussi envisager de faire une étude théorique sur la stabilité dans le but de garantir un certain niveau de performance de notre système. Nous voulons enfin, mener

des tests sur des problèmes réels, comme des problèmes d'emplois du temps, des problème d'ordonnancement de service au sol d'aéroport (*Airport Ground Sevice Scheduling Problems*), ou des problèmes d'ordonnancement dans des chantiers de construction.

Annexe A

Résultats des tests

Dans cette annexe, nous reportons les tableaux des résultats obtenus lors de nos expérimentations pour la résolution du RCPSP dynamique.

Nous rappelons que le symbole Na signifie que les valeurs utilisées sont obtenues par notre approche, et le symbole Ro correspond au ré-ordonnancement. P_0 représente le problème initial, et P_i le problème obtenu à la *ième* perturbation.

Pour chaque classe de problèmes et pour chaque perturbation, nous présentons :

- un tableau représentant la différence entre le temps T_{Na} (en seconde) nécessaire pour calculer une nouvelle solution par notre système et le temps T_{Ro} nécessaire en utilisant l'approche classique (ré-ordonnancement), puis le gain moyen en temps G_{temps} (*cf.* section 10.4.4 page 135). Nous reportons, dans le même tableau, la moyenne des temps nécessaires pour produire les solutions successives de la série des perturbations considérée, le temps minimal, puis le temps maximal. La première ligne concerne notre approche et la seconde à l'approche classique.

  Et enfin, un histogramme représentant la moyenne des temps, le temps minimal et le temps maximal;

- des tableaux représentant pour chaque problème et pour chaque perturbation les mesures de performance entre les solutions successives produites par notre approche (première ligne du tabeau) et celles produites par l'approche classique (deuxième ligne du tabeau). Dans ces tableaux, le symbole $\star$ signifie que la date de fin du projet (C_{max}) a augmenté ou a diminué.

  Puis quatre histogrammes représentant, pour chacune des mesures DIFF, POSI, SDIF et MDIF (*cf.* section 10.4.4 page 135), les valeurs obtenues par notre approche et par l'approche classique.

Dans les histogrammes, chaque problème est represenété par un rectangle de longueur égale à la valeur qui lui est associée (moyenne, nombre d'activités qui ont changé de date de début ...). Et chaque série de problèmes est représentée par des rectangles de même couleur. Ce qui nous permet de suivre l'évolution de chaque problème au cours du temps.

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série d'activités

Problèmes comportant 12 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		**moy.**	**min.**	**max.**
1.27	43%	1.04	46%	0.87	0.45	1.23
				1.64	1.23	1.97
2.46	40.3%	18.73	60.3%	4.64	1.28	10.3
				11.71	2.36	29.03
0.41	56.1%	-0.42	-0.7%	0.44	0.02	1.00
				0.43	0.3	0.58
-0.03	-0.9%	8.32	69.6%	1.20	0.55	1.66
				3.96	1.37	8.87
3.80	56.8%	1.07	41.8%	2.26	0.35	3.89
				3.88	0.35	6.34
5.57	60.2%	1.06	41.1%	3.16	1.52	5.82
				5.37	2.16	7.09
4.02	23.2%	9.83	44%	5.86	4.34	6.92
				10.48	6.33	14.17
5.12	64.4%	-3.36	13.3%	3.79	0.69	8.55
				4.38	2.14	5.81
3.34	34.5%	1.99	36.1%	3.13	2.09	4.42
				4.91	4.24	5.43
-0.01	-0.7%	-1.72	-63.8%	1.48	0.55	3.09
				0.90	0.55	1.37
moy.	37.7%		32.2%	43.7%		

TAB. A.1 – *Résultats pour le gain en temps pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

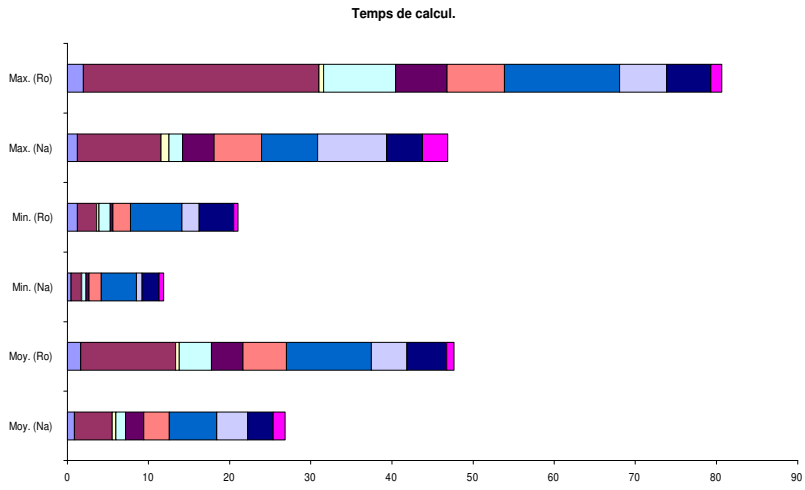


FIG. A.1 – *Représentation de quelques résultats du tableau A.1.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série d'activités

Problèmes comportant 22 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		**moy.**	**min.**	**max.**
-3.58	-8.7%	19.75	21.6%	571.83	85.1%	25.72	6.57	44.32
						172.72	6.57	616.15
19.22	36.7%	101.24	75.6%	573.18	89.6%	20.11	5.75	41.60
						193.52	9.90	614.78
273.35	60%	922.26	84.9%	674.88	88%	63.72	30.08	101.17
						531.34	101.17	952.34
107.43	45.4%	-99.48	1.8%	644.50	53.1%	143.74	38.53	293.69
						306.85	38.53	797.08
135.72	40.2%	12.74	31.6%	231.58	49.9%	95.04	59.54	119.19
						190.05	103.38	291.12
35.73	13%	222.18	43.4%	217.04	53.6%	102.51	74.79	131.91
						221.25	131.91	319.77
-1.83	-27%	-32.85	-173.3%	-8.93	-105.7%	21.21	3.38	46.10
						10.31	3.38	21.23
-5.10	-34.6%	-33.28	-208%	-23.10	-155.2%	25.26	2.85	44.24
						9.89	2.85	21.14
82.90	77.5%	129.62	86.5%	185.93	90%	10.97	8.94	13.76
						110.58	13.76	196.81
948.12	94.4%	1188.38	94.4%	1235.58	96%	34.25	11.01	69.94
						877.27	18.02	1258.32
moy.	29.7%		5.8%		34.4%	78.9%		

TAB. A.2 – *Résultats pour le gain en temps pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

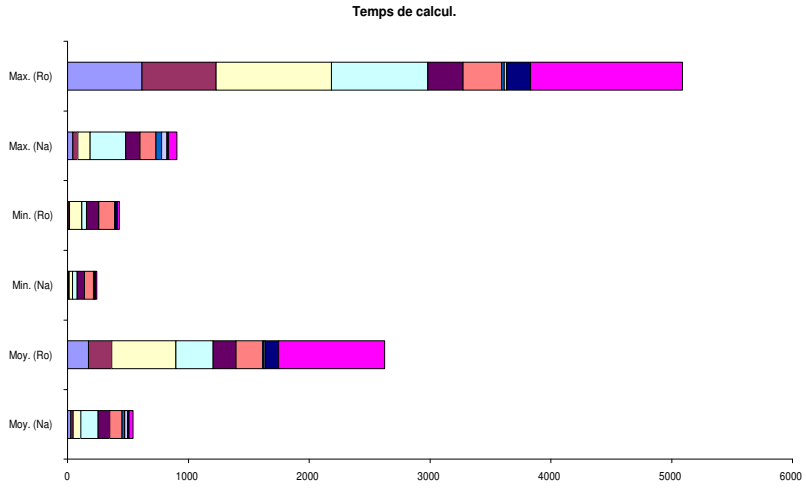


FIG. A.2 – *Représentation de quelques résultats du tableau A.2.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série d'activités

Problèmes comportant 32 activités et une ressource

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
128.94	41.6%	111.05	48.6%	179.87	56.6%	-363.09	6.3%	167.55	64.79	51.68
								178.91	116.09	247.10
457.22	98.7%	466.28	99.3%	2.52	98.7%	11.99	98.7%	2.36	0.04	5.91
								189.96	5.72	466.32
5.36	53.2%	5.25	65.6%	7.06	76%	1400.03	99.6%	1.12	0.03	4.65
								284.66	4.65	1400.07
7.27	50.6%	8.27	68.5%	10.38	78.3%	8.43	75.3%	2.25	0.05	7.02
								9.12	7.02	12.51
215.13	91.7%	267.17	96.2%	292.06	97.5%	364.38	97.4%	6.07	0.06	12.69
								233.82	6.70	375.18
6.67	53.1%	9.96	73.6%	8.02	66.1%	-21.33	6.3%	9.83	0.03	36.51
								10.49	5.86	15.18
12.47	57.7%	15.56	75.3%	696.06	98.3%	1575.28	96.8%	14.88	0.05	62.15
								474.75	9.06	1637.43
3.82	30%	-0.12	18.2%	7.71	40.7%	-35.67	-66.1%	12.18	0.05	44.32
								7.33	5.97	8.65
6.82	55%	7.79	72.1%	8.86	80.3%	11.78	85.9%	1.15	0.04	5.47
								8.20	5.47	11.82
9.18	52%	9.14	68.3%	12.46	78.2%	-47.22	-32.5%	13.39	0.03	58.41
								10.11	8.43	12.54
moy.	58.3%		68.5%		77%		46.7%	83.6%		

TAB. A.3 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et une ressource. Le cas où une série d'activités est ajoutée.*

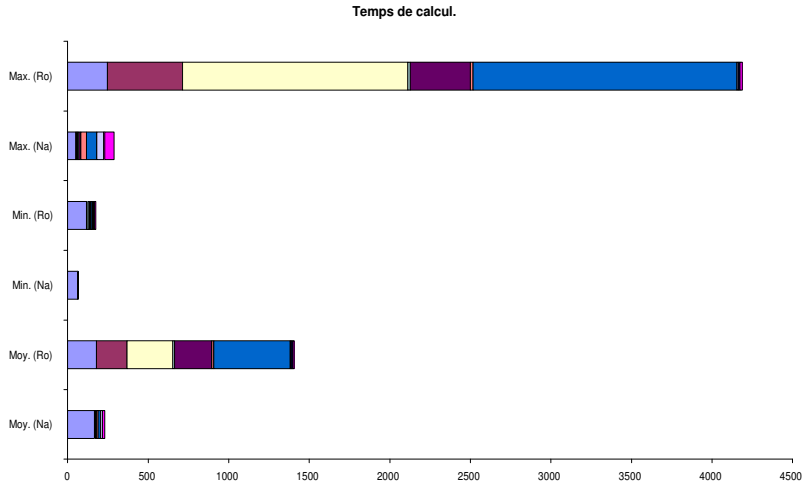


FIG. A.3 – *Représentation de quelques résultats du tableau A.3.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série d'activités

Problèmes comportant 32 activités et 2 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
267.71	91.2%	761.56	97.1%	857.07	96.9%	708.07	97.4%	13.32	4.44	28.83
								532.20	7.80	885.90
11.00	55.2%	591.13	93.8%	616.23	92.8%	779.80	95.5%	18.82	0.08	54.53
								418.46	8.82	779.99
150.42	86.9%	-7.43	72.3%	3.37	69.8%	-105.53	4.9%	157.78	0.19	725.63
								165.95	11.97	620.10
199.06	41.3%	292.44	54.3%	346.40	62%	684.18	66.7%	151.87	90.90	246.98
								456.28	191.47	931.16
-35.13	-5.6%	1403.52	65.6%	1538.31	78.6%	1299.90	83.5%	165.20	34.79	444.31
								1006.55	214.66	1613.71
9.92	57.5%	14.79	76.9%	29.14	87.8%	29.64	86.5%	2.60	0.07	7.20
								19.29	7.20	35.18
3.64	21.1%	10.60	51%	-44.75	-64.2%	-34.75	-78.9%	29.58	1.30	69.90
								16.53	7.55	35.15
20.39	16%	991.11	81.5%	186.26	76.4%	375.89	79.7%	80.07	31.61	140.07
								394.80	55.99	1113.54
-43.40	-280.9%	-8.76	-140.1%	1762.57	92.1%	1265.04	95.3%	29.15	0.11	56.28
								624.24	7.59	1818.85
8.08	24.5%	907.96	90.5%	1169.14	95.6%	1897.92	97.6%	19.15	0.17	70.56
								815.77	11.93	1898.09
moy.	10.7%		57.8%		68.7%		62.8%	84.9%		

TAB. A.4 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et 2 ressources. Le cas où une série d'activités est ajoutée.*

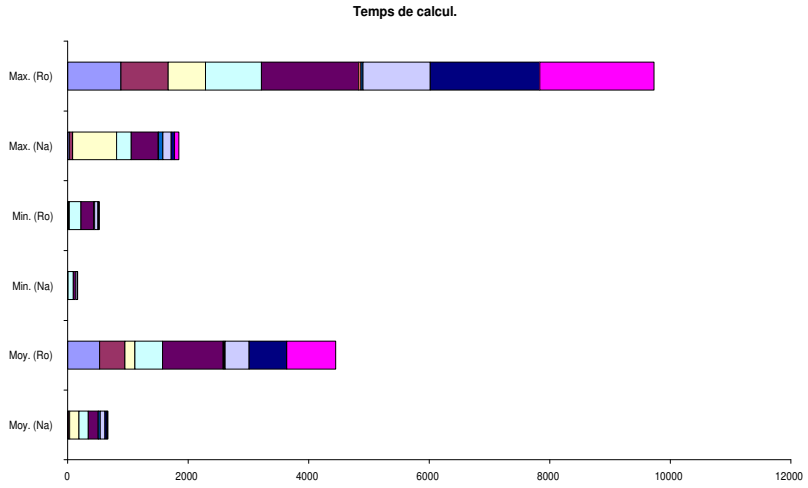


FIG. A.4 – *Représentation de quelques résultats du tableau A.4.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série d'activités

Problèmes comportant 32 activités et 3 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
1670.55	93%	1815.67	96.5%	1798.12	97.6%	1232.49	96.3%	49.35	0.21	121.76
								1352.72	80.70	1815.92
750.94	95.2%	1352.25	98.2%	1897.72	98.8%	1636.33	97.5%	28.65	0.32	98.38
								1156.10	14.55	1904.63
63.24	12.7%	57.79	17.6%	213.70	36.1%	203.24	45.1%	130.57	27.85	364.21
								238.16	131.04	364.21
455.73	86.8%	661.99	94.1%	479.85	95.8%	609.86	95.5%	20.47	0.19	49.41
								461.95	19.64	662.40
12.32	52.8%	10.84	52.6%	26.87	70.4%	269.82	93.7%	4.25	0.15	10.78
								68.22	10.78	270.13
17.59	56.8%	184.50	87.2%	178.34	89.4%	247.43	90.2%	13.53	0.19	22.83
								139.10	13.14	270.26
10.32	43.8%	-1.19	15.9%	5.30	16.8%	44.16	30.6%	26.53	3.10	61.34
								38.25	10.09	105.50
17.74	42.9%	19.81	27.2%	154.36	65.5%	105.27	74.6%	20.14	0.07	76.87
								79.58	16.68	154.61
121.02	67%	106.21	79.1%	180.16	87%	169.90	74.3%	39.27	0.32	138.14
								155.18	59.33	308.04
156.38	90.6%	38.66	92.3%	154.14	89%	245.30	84.8%	21.23	0.08	63.26
								140.12	15.98	308.56
moy.	64.1%		66%		74.6%		78.2%	90.5%		

TAB. A.5 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et 3 ressources. Le cas où une série d'activités est ajoutée.*

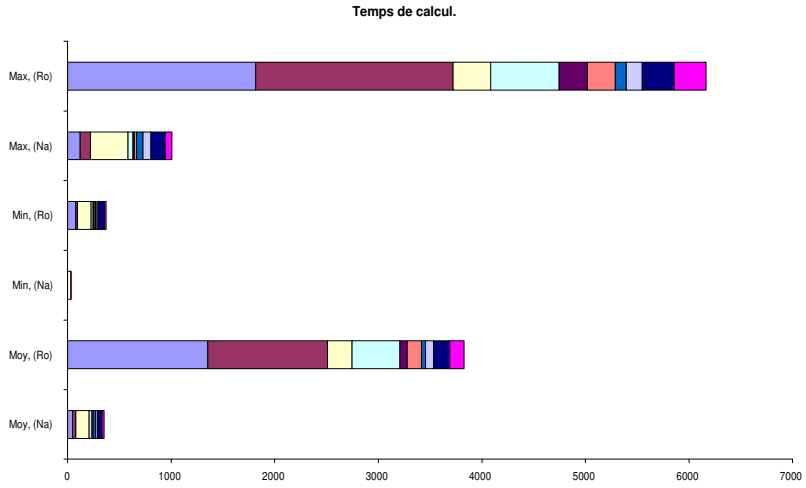


FIG. A.5 – *Représentation de quelques résultats du tableau A.5.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série d'activités

Problèmes comportant 32 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
21.31	49.5%	16.93	63.4%	21.96	72.9%	31.29	73.7%	6.50	0.21	21.52
								24.79	17.26	41.43
18.57	51.1%	7.39	50.5%	14.44	61.1%	19.58	69.8%	5.17	0.21	17.47
								17.17	14.69	19.79
19.74	65.6%	11.70	68.7%	14.84	76%	19.62	81.7%	2.95	0.16	10.18
								16.13	10.18	19.90
11.09	33.3%	-19.78	-18.1%	-3.14	-19.6%	246.37	76.4%	14.45	0.16	34.36
								61.36	12.47	246.53
17.35	65.7%	14.42	77.6%	3.11	65.3%	246.31	93.7%	3.73	0.10	9.37
								59.96	8.86	246.47
45.29	75.9%	49.35	86.7%	46.80	86%	69.41	87.1%	6.19	0.13	14.18
								48.36	14.18	77.42
22.23	46.3%	18.81	55.3%	33.18	68.9%	446.93	82.1%	22.70	0.18	80.05
								126.93	22.41	526.98
55.94	57%	59.09	73%	-93.10	11.5%	41.65	26.7%	34.90	0.04	125.34
								47.61	32.24	59.44
21.56	51.5%	755.25	97.4%	847.77	98.7%	1244.10	99.2%	4.20	0.20	20.02
								577.94	20.02	1244.30
103.83	73.9%	128.31	80.9%	154.36	86.3%	561.58	91.7%	17.14	6.48	24.50
								206.76	21.75	586.08
moy.	57%		63.5%		60.7%		78.2%	90%		

TAB. A.6 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

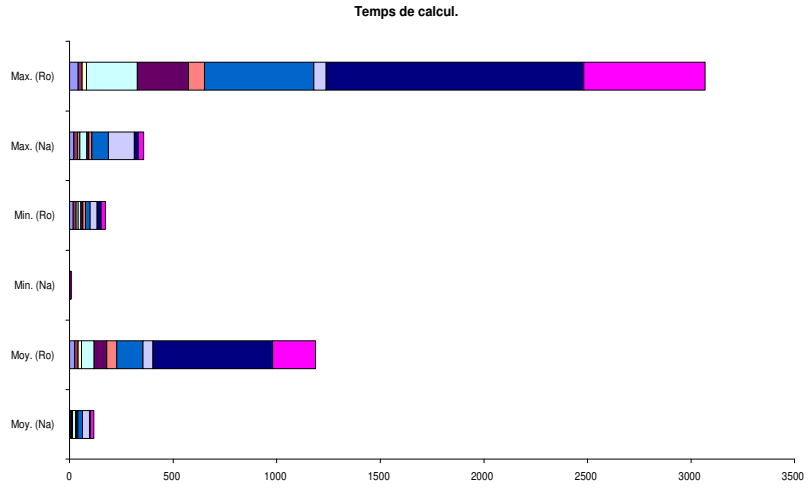


FIG. A.6 – *Représentation de quelques résultats du tableau A.6.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série d'activités

Problèmes comportant 12 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		**moy.**	**min.**	**max.**
-1.06	-28.4%	-0.81	-38.4%	2.24	1.94	2.84
				1.62	1.14	1.94
11.00	21.7%	-0.28	19.5%	14.66	4.30	32.53
				18.23	4.02	32.53
-2.95	-212.2%	-1.47	-195.5%	2.22	0.58	3.76
				0.75	0.58	0.87
2.61	14.9%	3.75	27.7%	5.53	1.69	11.71
				7.65	5.44	11.71
0.75	7%	-1.23	-4%	3.78	1.51	6.66
				3.62	0.28	6.66
-12.24	-160.4%	-2.00	-167.5%	7.58	2.87	13.47
				2.83	0.87	6.40
-8.75	-35.9%	-2.98	-42%	13.20	6.52	18.77
				9.29	3.54	18.77
0.13	2.4%	-2.61	-41.8%	2.80	2.07	3.32
				1.97	0.71	3.14
-6.01	-74.1%	-0.67	-71.9%	5.32	1.84	7.64
				3.09	1.17	6.48
-1.15	-45%	-2.76	-139.1%	2.24	1.68	3.02
				0.93	0.26	1.68
moy.	-51%		-65.3%	-19.1%		

TAB. A.7 – *Résultats pour le gain en temps pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série d'activités est retirée.*

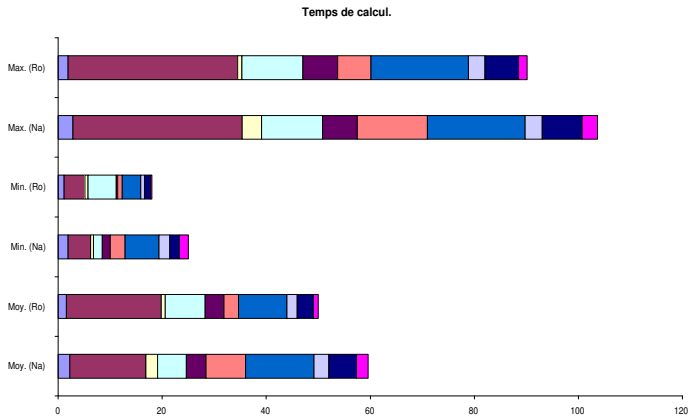


FIG. A.7 – *Représentation de quelques résultats du tableau A.7.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série d'activités

Problèmes comportant 22 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		**moy.**	**min.**	**max.**
1.58	15.8%	62.05	46.7%	6.32	44.3%	21.93	3.50	64.15
						39.42	4.87	126.20
3.98	0.7%	-25.57	-3.7%	25.93	0.6%	155.78	20.61	518.97
						156.87	21.53	518.97
110.12	62.3%	-4.03	58.8%	-0.46	57.7%	19.33	3.15	33.97
						45.74	2.69	142.61
-154.29	-637.8%	-4.19	-242.8%	-6.52	-232.1%	59.01	12.32	163.95
						17.76	5.80	41.07
81.57	72.9%	46.64	68.6%	232.97	43.5%	117.14	5.33	410.05
						207.44	5.33	643.02
142.81	29.7%	24.46	28.4%	64.76	34.8%	108.67	13.4	187.16
						166.68	78.16	329.97
366.21	77.3%	310.96	83.9%	155.26	85.2%	36.02	14.61	73.94
						244.12	33.40	440.15
72.91	72.1%	22.20	68.9%	46.00	79.9%	8.87	3.54	22.35
						44.14	22.35	78.68
130.94	91.5%	109.77	94.3%	27.63	88.8%	8.44	2.47	19.23
						75.52	5.05	137.95
-19.76	-13.6%	24.80	2.8%	20.99	12.9%	43.91	3.17	97.56
						50.41	24.16	97.56
moy.	-22.9%		20.5%		21.5%	44.7%		

TAB. A.8 – *Résultats pour le gain en temps pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série d'activités est retirée.*

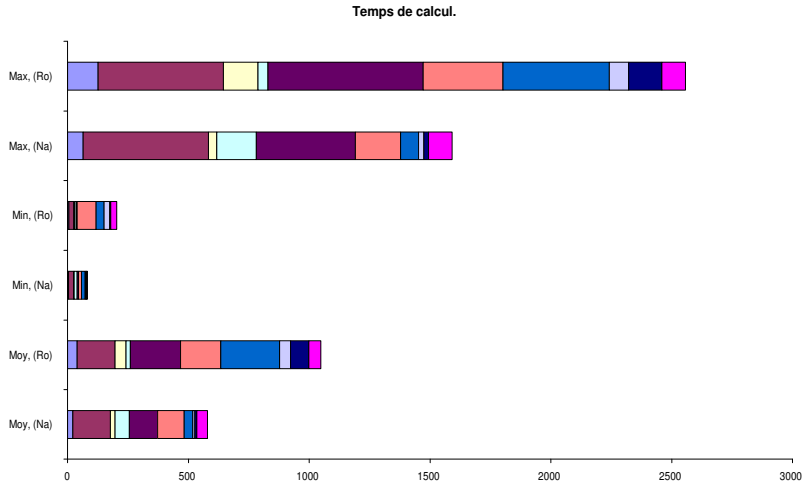


FIG. A.8 – *Représentation de quelques résultats du tableau A.8.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série d'activités

Problèmes comportant 32 activités et une ressource

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
217.82	80.9%	423.24	90.9%	120.43	91.1%	116.00	91.5%	16.10	6.99	35.01
								191.60	35.01	435.72
-10.01	-2.8%	-114.38	-28.5%	-204.53	-65.4%	-92.37	-78.6%	191.39	125.22	271.38
								107.13	32.85	198.72
706.76	97.1%	447.15	92.7%	-22.86	90.3%	-457.21	53.5%	116.68	8.07	462.37
								251.44	5.16	714.83
-30.00	-185%	2.09	-119%	0.67	-92.7%	-3.89	-89.6%	13.17	5.14	38.50
								6.94	5.36	8.50
-2.29	-9.2%	-0.59	-8.6%	-23.40	-62.7%	-49.08	-151.5%	25.02	9.09	56.96
								9.94	7.88	12.79
174.34	20.6%	227.07	37.5%	256.24	49.3%	236.14	56.7%	136.18	6.36	648.16
								314.94	185.31	648.16
116.27	39.1%	307.21	43.7%	355.84	48.3%	197.78	50.1%	415.75	49.39	1715.85
								833.37	257.55	1715.85
125.01	73.3%	50.25	76.6%	15.86	75.9%	7.29	73.6%	13.33	6.21	37.49
								53.01	13.50	133.00
88.93	61.8%	32.37	65.6%	18.21	66.3%	91.53	75.2%	15.92	5.21	45.20
								61.4	25.64	98.67
-8.86	-41.6%	-5.40	-44.5%	-27.25	-102.6%	-1.58	-86.8%	18.54	10.76	35.68
								9.92	8.43	11.97
moy.	13.4%		20.6%		0.9%		-0.5%	47.7%		

TAB. A.9 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et une ressource. Le cas où une série d'activités est retirée.*

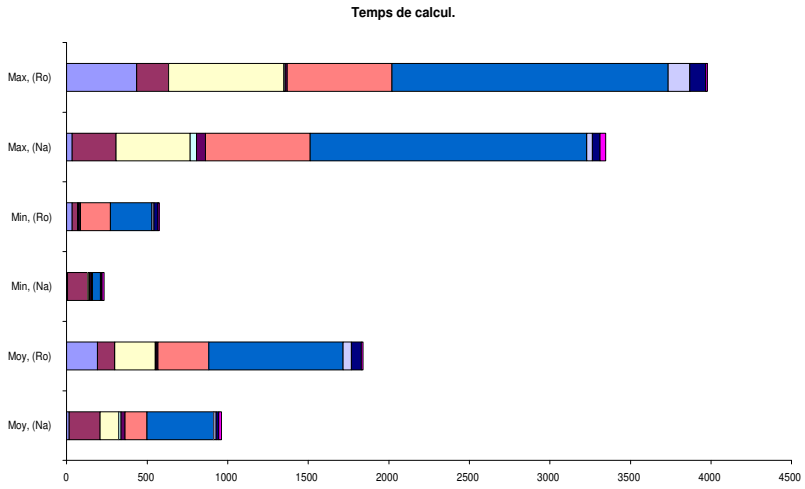


FIG. A.9 – *Représentation de quelques résultats du tableau A.9.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série d'activités

Problèmes comportant 32 activités et 2 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
869.87	95.1%	702.04	96.8%	714.27	97.3%	503.82	97.5%	14.14	7.01	23.81
								572.14	20.07	893.68
737.98	93.9%	168.48	88.7%	220.94	90%	3.21	89.7%	25.73	4.48	67.14
								251.85	7.69	748.86
53.19	44.5%	42.37	47.6%	548.91	82%	593.23	88.2%	33.05	24.21	41.57
								280.59	24.53	617.44
142.46	17.3%	43.45	17.2%	-306.79	-9.9%	54.58	-4.6%	295.77	147.60	440.50
								282.51	133.71	523.19
-88.53	-15.1%	226.41	15%	237.10	30.4%	103.23	33.7%	187.99	78.63	460.10
								283.64	185.99	371.57
-57.16	-85.2%	-53.18	-128.6%	-47.59	-161.2%	-19.70	-164.8%	57.07	29.51	76.88
								21.55	9.81	47.35
120.47	78.4%	38.27	75.3%	33.42	78.2%	27.63	76.9%	13.19	1.55	18.81
								57.14	14.67	138.89
384.31	98%	359.73	98.7%	125.65	97.4%	-238.16	67.3%	61.16	0.01	283.15
								187.47	7.63	384.15
-56.81	-148.4%	-53.03	-219.4%	306.81	37.4%	-126.48	8.7%	146.88	22.31	405.19
								160.98	11.77	476.15
394.62	64.2%	-49.76	53.2%	-20.99	48.4%	0.14	47.5%	71.53	13.03	151.46
								136.34	13.17	463.08
moy.	24.2%		14.4%		38.9%		34%	59.4%		

TAB. A.10 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et 2 ressources. Le cas où une série d'activités est retirée.*

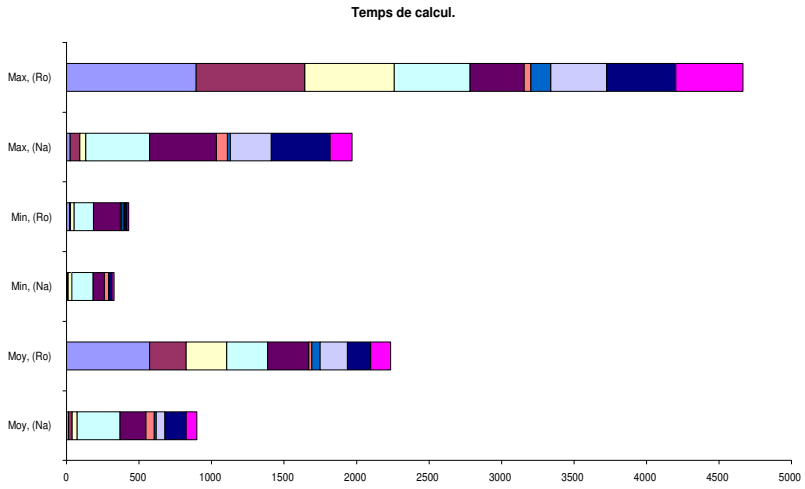


FIG. A.10 – *Représentation de quelques résultats du tableau A.10.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série d'activités

Problèmes comportant 32 activités et 3 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
-0.12	-0%	16.51	5%	24.55	11.6%	0.59	11.3%	65.04	5.20	250.24
								73.34	13.83	250.12
290.59	77%	224.58	81.2%	131.74	80.2%	59.30	79.6%	36.07	20.75	55.06
								177.32	31.68	345.65
731.24	93.2%	303.66	94%	249.32	91.7%	173.25	90.3%	31.13	12.14	49.60
								322.62	31.49	753.10
574.84	94.1%	-979.99	-34.8%	-53.53	-38.3%	-66.71	-43.1%	348.33	3.16	1531.60
								243.26	19.75	578.00
59.28	22.7%	38.74	16.1%	13.64	14.3%	150.81	27.6%	137.48	23.09	307.79
								189.98	26.39	346.53
336.43	89.1%	154.69	92%	0.40	90.2%	3.50	89.5%	11.49	1.62	30.12
								110.50	7.80	347.36
312.11	68.5%	313.13	75%	191.77	75.7%	-46.09	70.6%	64.08	36.53	106.90
								218.26	12.76	419.01
180.96	75.3%	340.78	57.8%	343.59	64.3%	284.30	67.2%	112.00	27.18	321.05
								341.93	32.17	661.83
369.11	87.4%	141.55	85.6%	38.38	84.4%	17.50	83.9%	21.62	7.17	32.38
								134.93	24.67	390.67
148.26	47.9%	92.10	51.1%	-191.06	7.1%	84.70	15.6%	144.98	68.59	411.32
								171.78	69.10	239.80
moy.	65.5%		52.3%		48.1%		49.2%	51.8%		

TAB. A.11 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et 3 ressources. Le cas où une série d'activités est retirée.*

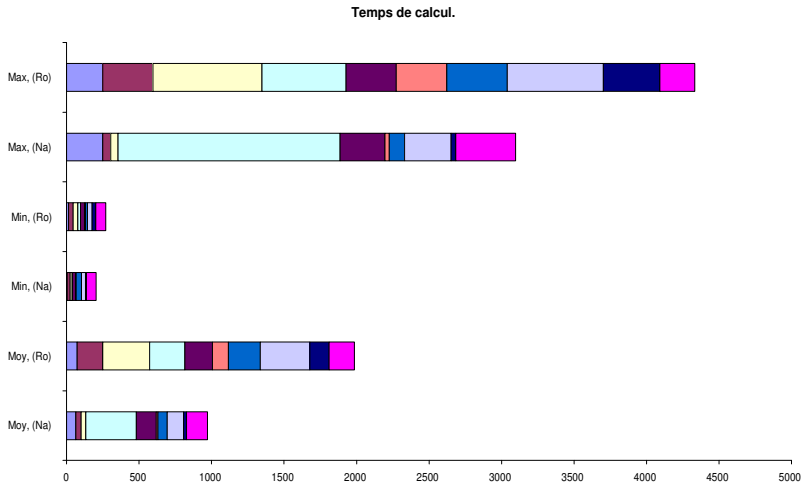


FIG. A.11 – *Représentation de quelques résultats du tableau A.11.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série d'activités

Problèmes comportant 32 activités et 4 ressources

$P_0 \to P_1$		$P_1 \to P_2$		$P_2 \to P_3$		$P_3 \to P_4$		**moy.**	**min.**	**max.**
526.04	88.5%	430.10	92.6%	-1272.82	-27%	-261.61	-45.8%	367.75	7.69	1411.55
								252.09	41.61	552.62
407.32	77.6%	318.09	81.9%	70.94	81.3%	48.56	81.5%	38.29	8.60	77.39
								207.27	39.68	484.71
520.14	80.9%	-45.13	71.2%	-19.78	65.7%	0.58	63.2%	53.06	28.47	82.30
								144.22	23.80	602.44
-99.50	-280.4%	-35.88	-259.5%	4.40	-180.2%	3.65	-148.1%	42.65	9.62	115.52
								17.18	13.27	20.50
361.82	92.4%	333.28	95.7%	248.70	92.3%	164.54	91.3%	21.01	1.52	47.08
								242.68	15.44	376.06
53.22	65.4%	276.37	86.8%	-0.04	82.7%	-177.18	31.1%	67.40	11.77	268.18
								97.88	16.32	298.30
-8.27	-12.6%	-46.66	-63.5%	-9.03	-58.9%	15.28	-36%	36.72	11.17	67.62
								26.98	20.96	39.16
401.04	80.2%	804.10	86.3%	787.13	89.2%	325.20	89.2%	55.64	24.67	92.27
								519.14	24.67	896.37
43.41	35.1%	113.94	59.9%	74.42	64.4%	24.90	62.6%	30.65	22.46	48.22
								81.98	48.22	139.17
763.26	83.4%	379.02	73.6%	275.26	74%	248.09	74.6%	113.09	27.15	257.01
								446.22	27.15	887.87
moy.	31%		32.5%		28.3%		26.3%	59.4%		

TAB. A.12 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités et 4 ressources. Le cas où une série d'activités est retirée.*

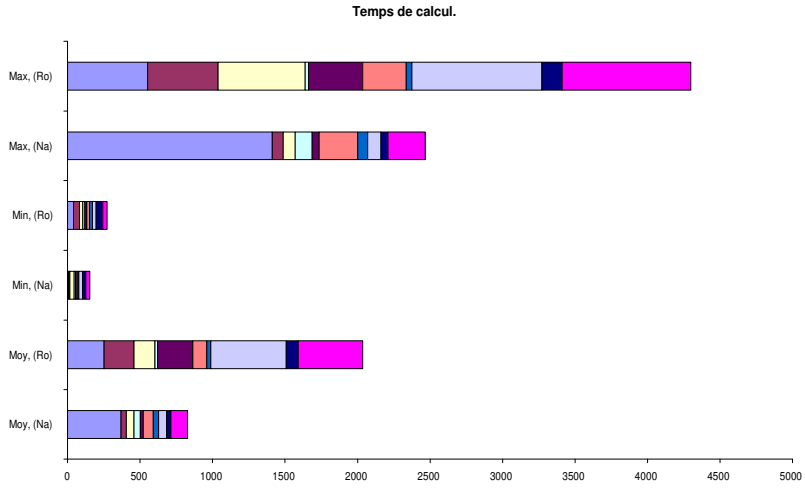


FIG. A.12 – *Représentation de quelques résultats du tableau A.12.*

Résultats obtenus pour le temps de calcul CPU
Ajout d'une série de contraintes de précédence
Problèmes comportant 12 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
-0.55	-4.5%	18.03	46.1%	19.28	63.9%	-10.43	42%	7.25	0.36	15.54
								12.52	4.67	25.69
4.53	35.2%	-0.85	27.3%	-5.25	-11.2%	4.54	13.1%	3.94	1.44	5.81
								4.53	0.56	8.66
2.50	9.7%	6.16	24.9%	-1.17	20.7%	-0.58	17.8%	6.38	2.48	16.69
								7.76	1.31	16.69
4.72	34.6%	4.05	41.5%	-1.85	31.9%	-1.34	25.2%	3.30	1.79	6.37
								4.41	0.45	7.50
41.44	60.1%	0.54	50.9%	4.33	48.6%	12.48	50.7%	11.39	8.13	15.94
								23.15	11.56	57.38
16.67	34%	-3.29	21.2%	-35.00	15.1%	-21.68	-17.2%	15.94	6.01	23.85
								13.60	2.17	30.59
2.89	16.4%	-4.46	-6.8%	-2.26	-1.6%	-6.68	-43.6%	6.91	2.93	9.71
								4.81	0.53	9.15
6.11	29.5%	5.88	37.4%	5.35	42.7%	6.85	47.8%	5.27	3.14	9.61
								10.11	8.53	11.33
0.40	38%	0.74	-47.5%	1.06	61.4%	-0.87	32.4%	0.55	0.12	1.39
								0.82	0.50	1.35
moy.	28.1%		21.6%		30.1%		18.7%	25.4%		

TAB. A.13 – *Résultats pour le gain en temps pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes de précédence est ajoutée.*

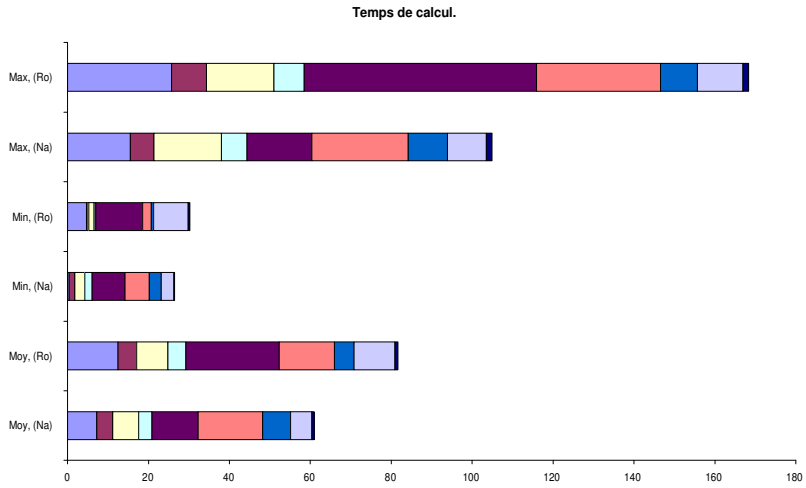


FIG. A.13 – *Représentation de quelques résultats du tableau A.13.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série de contraintes de précédence

Problèmes comportant 22 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		**moy.**	**min.**	**max.**
178.95	82%	389.07	79.2%	49.60	13.71	109.61
				238.94	25.50	498.68
-4.38	-1.7%	87.14	19.35%	114.94	90.77	154.88
				142.52	99.17	177.91
257.43	90.9%	-19.19	37.3%	133.20	10.53	373.88
				212.61	15.19	354.69
28.85	35.2%	-9.12	15.7%	35.06	11.93	52.25
				41.64	40.78	43.13
15.26	2.6%	7.08	3.1%	229.19	133.14	394.53
				236.63	140.22	394.53
18.54	45.1%	12.55	49.3%	10.64	5.79	16.74
				21.00	5.79	35.28
2.65	25.2%	0.07	15.5%	4.91	2.48	6.87
				5.81	5.13	6.94
-24.45	-15.3%	22.72	-0.4%	124.60	34.61	189.73
				124.02	34.61	212.45
46.82	12%	12.67	13.1%	130.57	50.17	197.68
				150.40	62.84	197.68
93.72	27.8%	37.37	24.8%	131.89	68.50	174.63
				175.59	162.22	189.92
moy.	30.0%		25.6%	28.5%		

TAB. A.14 – *Résultats pour le gain en temps pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes de précédence est ajoutée.*

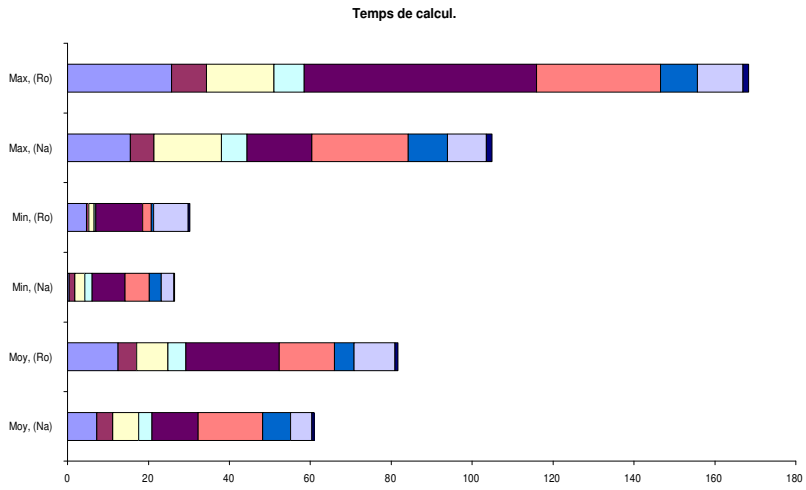


FIG. A.14 – *Représentation de quelques résultats du tableau A.13.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série de contraintes de précédence

Problèmes comportant 32 activités

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		**moy.**	**min.**	**max.**
-61.22	-18.8%	-21.33	-18.6%	-72.44	-26.6%	184.07	140.66	214.95
						145.32	119.33	170.47
-237.20	-49.3%	-67.21	-50%	48.29	-34.5%	249.20	83.75	383.07
						185.17	128.30	334.48
55.89	63.9%	72.93	78.9%	32.51	69.5%	17.69	2.76	36.47
						58.02	9.39	78.04
100.06	76.9%	106.06	82%	98.36	82%	16.63	12.16	21.44
						92.75	12.16	121.22
-131.19	-35.9%	-92.11	-58.1%	-2.68	-55.1%	158.93	28.68	321.29
						102.44	18.57	190.10
86.02	68.1%	75.82	58.5%	20.55	46.7%	51.87	15.58	93.02
						97.46	15.58	150.13
-86.71	-32.7%	61.46	-5.4%	20.28	-0.7%	158.03	128.74	223.05
						156.79	128.74	194.38
143.78	76.9%	50.87	65.9%	82.45	70%	29.67	13.82	57.49
						98.94	13.82	173.11
-9.48	-4.2%	-84.74	-28.6%	-18.52	-25.7%	137.67	108.20	190.93
						109.48	106.19	115.04
82.30	72.6%	190.28	85.2%	34.43	75.8%	24.49	12.31	50.90
						101.24	12.31	206.31
13.80	2.6%	139.77	20.8%	189.42	34.9%	159.40	55.28	324.19
						245.15	199.62	324.19
74.35	41.4%	-80.28	-2.1%	12.22	1.9%	78.98	33.69	177.33
						80.55	45.91	132.78
32.76	36.6%	20.08	43.8%	46.08	55.7%	19.62	10.86	38.29
						44.35	18.22	71.05
38.21	32.3%	-30.88	5.1%	-46.85	-23.5%	51.90	24.23	72.83
						42.02	23.98	62.44
-20.49	-19.5%	6.42	-5.9%	145.01	32.6%	67.61	18.90	126.05
						100.35	35.09	163.91
106.27	54.3%	41.67	53%	26.14	50.2%	43.10	41.61	45.75
						86.62	43.39	152.02
13.96	19%	-16.34	-2.2%	-10.77	-9.5%	37.59	21.38	48.31
						34.30	31.97	37.93
58.51	43.3%	-101.41	-21.3%	-171.37	-91.8%	111.86	9.92	204.04
						58.29	32.67	68.43
-106.94	-43%	20.50	-23.6%	-148.33	-53%	169.37	96.01	230.51
						110.68	77.97	124.69
375.69	29.5%	215.03	34.9%	129.95	38.6%	285.85	43.95	651.99
						466.02	173.90	651.99
moy.	41.4%		31.2%		23.7%	15%		

TAB. A.15 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est ajoutée.*

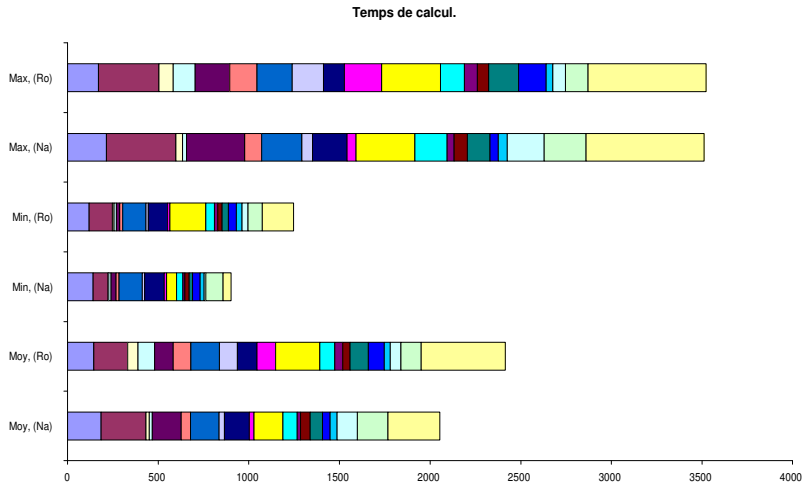


FIG. A.15 – *Représentation de quelques résultats du tableau A.15.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série de contraintes de précédence
Problèmes comportant 12 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		**moy.**	**min.**	**max.**
0.84	14.8%	3.16	29.2%	3.22	2.11	4.86
				4.56	2.11	8.02
-18.28	-195.7%	-6.50	-148.8%	13.81	1.63	25.99
				5.55	1.63	7.71
-0.76	-60.3%	0.02	-38.7%	0.88	0.62	1.40
				0.63	0.62	0.65
-0.16	-3.9%	18.25	61.1%	3.83	1.29	7.23
				9.86	1.29	25.48
-3.95	-23.3%	-0.05	-11.2%	13.16	0.53	20.34
				11.83	0.53	18.57
517.75	96%	62.86	88.8%	24.36	7.30	51.69
				217.90	7.30	531.86
-1.25	-6.8%	-3.34	-23.9%	7.90	4.28	11.22
				6.37	0.94	11.22
9.54	70%	-5.11	20.3%	5.75	1.95	13.18
				7.22	2.12	11.49
-3.32	-26.9%	1.38	-8.5%	8.21	0.77	14.88
				7.56	0.77	11.56
25.57	93.1%	9.63	78.8%	3.15	0.57	7.58
				14.89	0.57	26.89
moy.	4.3%		4.7%	70.5%		

TAB. A.16 – *Résultats pour le gain en temps pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes de précédence est retirée.*

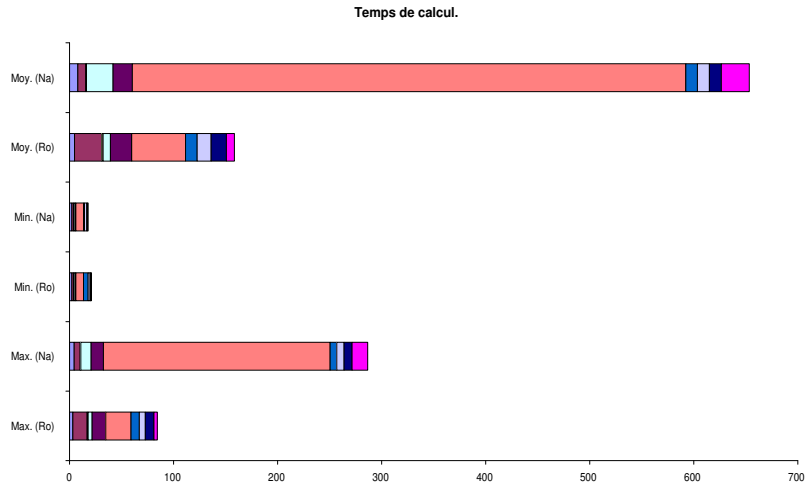


FIG. A.16 – *Représentation de quelques résultats du tableau A.16.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série de contraintes de précédence

Problèmes comportant 22 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		**moy.**	**min.**	**max.**
-35.07	-14.5%	11.17	-8.7%	14.43	-3%	79.26	18.71	222.06
						76.90	19.35	222.06
26.25	23.2%	-29.04	-1.9%	-38.27	-22.2%	56.33	32.94	80.48
						46.07	28.98	80.15
6.61	18.8%	19.21	43.5%	-2.25	37.8%	9.65	4.88	21.89
						15.55	3.02	24.09
59.14	42.6%	52.14	34.8%	-17.10	24.3%	96.51	32.86	180.35
						72.97	32.86	128.21
26.75	10%	9.83	8.5%	-158.64	-21.1%	174.99	110.52	311.03
						144.47	128.91	159.33
1.06	2.5%	1.12	4.7%	2.24	8.5%	11.76	3.55	22.03
						12.87	4.86	22.03
2.11	6%	2.40	10.3%	-1.26	6.7%	11.25	6.12	21.93
						12.06	4.86	21.93
6.38	47.9%	-18.18	-53.3%	-8.41	-65.5%	12.75	1.97	27.01
						7.70	4.95	8.83
-106.93	-109.2%	2.77	-64.7%	-198.93	-152.1%	125.58	33.76	237.41
						49.81	33.76	64.15
-11.52	-4.3%	4.62	-1.6%	-189.06	-34.7%	189.88	131.65	323.24
						140.89	131.65	161.55
moy.	2.3%		-2.8%		-22.1%	-32.5%		

TAB. A.17 – *Résultats pour le gain en temps pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes de précédence est retirée.*

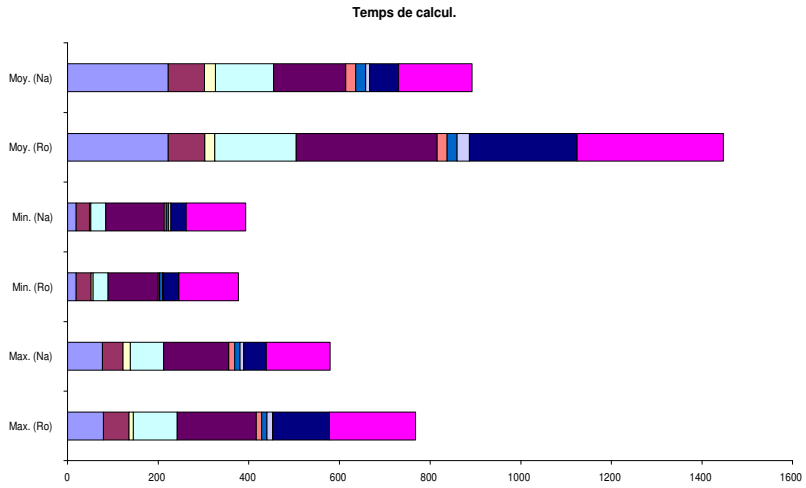


FIG. A.17 – *Représentation de quelques résultats du tableau A.17.*

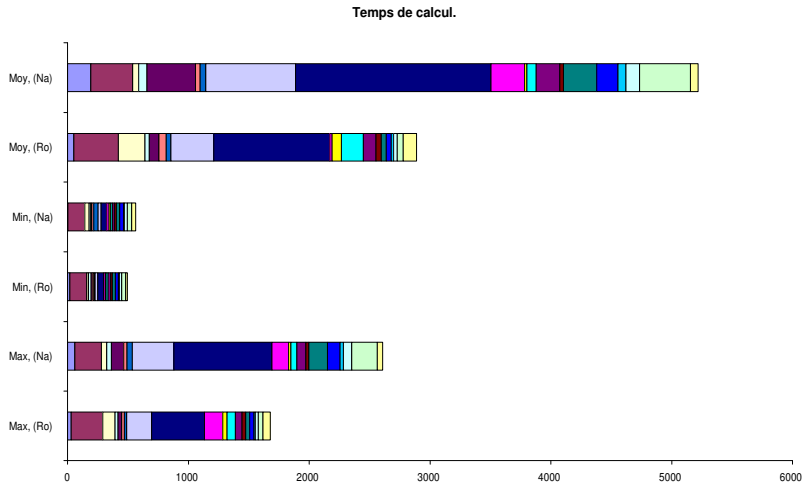


FIG. A.18 – *Représentation de quelques résultats du tableau A.18.*

Résultats obtenus pour le temps de calcul CPU

Retrait d'une série de contraintes de précédence
Problèmes comportant 32 activités

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
56.29	57.8%	169.28	78%	-29.12	65.1%	-49.01	48.5%	31.29	18.95	51.25
								60.78	2.24	191.75
-102.63	-34.6%	136.04	5.2%	-69.40	-4%	-177.50	-19.5%	260.68	140.22	368.96
								217.98	140.22	345.70
-46.88	-51.5%	35.77	-7.8%	-169.66	-93%	-100.19	-121.8%	102.29	15.35	221.76
								46.10	36.31	52.10
44.96	52.8%	-3.93	38.9%	-10.48	25.4%	31.59	33.1%	25.04	19.62	36.13
								37.47	14.49	67.72
7.89	20.8%	49.12	56.2%	-7.36	45.7%	324.59	73.1%	27.42	14.36	78.28
								102.27	7.17	402.87
15.66	32.7%	-22.73	-8.3%	13.60	5.8%	13.95	14.8%	23.49	11.63	60.11
								27.58	16.24	37.38
26.13	34.4%	17.79	35.6%	39.22	49%	35.31	54.9%	19.41	7.17	39.60
								43.10	36.16	47.37
41.43	45.2%	-72.29	-43.1%	255.70	22.8%	463.70	39.9%	207.08	24.21	354.90
								344.79	24.21	741.79
-317.80	-329.4%	115.65	-17.2%	954.19	30.8%	1124.45	46.2%	435.34	49.11	960.00
								810.64	47.34	1618.33
55.08	61.1%	262.58	86.6%	203.14	88.8%	102.39	89%	153.74	11.65	18.34
								140.01	18.34	276.68
-8.44	-25.6%	-59.55	-135.7%	-25.44	-142.1%	11.50	-96.5%	33.35	7.61	76.70
								16.96	15.63	19.11
-48.10	-60.8%	-3.52	-53.9%	61.04	5.4%	-108.98	-40.5%	69.05	16.08	181.95
								49.13	11.63	77.12
159.28	73.6%	-55.28	39%	-38.15	23.1%	31.17	25.9%	55.42	20.34	105.37
								74.82	17.04	195.89
10.41	20.9%	-18.55	-11.7%	-7.36	-17.8%	-13.90	-25.4%	29.01	13.32	42.71
								23.13	17.67	28.81
192.64	77.4%	109.68	75.2%	65.23	72.8%	238.92	77.6%	34.86	25.84	43.45
								156.15	25.84	276.45
110.39	63.7%	70.09	64%	143.64	71%	32.40	67.5%	34.31	27.04	39.24
								105.61	35.74	174.50
59.92	79.5%	33.79	78.9%	-6.37	67.4%	-13.27	55%	12.11	7.64	18.48
								26.92	5.21	67.56
69.17	60.2%	94.41	72%	-4.09	62.9%	57.78	63.2%	25.19	17.84	31.99
								68.65	22.78	112.25
165.81	66.1%	383.25	81.6%	128.33	80.3%	181.93	81.2%	39.75	33.62	47.85
								211.61	37.14	421.26
-76.43	-118.2%	-24.26	-97.1%	-9.64	-65.8%	42.79	-30.1%	58.28	13.51	109.50
								44.77	31.54	63.90
moy.	12.6%		33.6%		30.4%		43.6%	35.7%		

TAB. A.18 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est retirée.*

Résultats obtenus pour la stabilté
Ajout d'une série d'activités
Problèmes comportant 12 activités et 4 ressources

Diff.			Posi.		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
4	3	2	3	4	2
0	0	0	0	0	0
0	7⋆	6⋆	0	0	0⋆
0	7	6	0	3	3
0	2⋆	1⋆	0	6⋆	6⋆
1	1	1	0	0	0
4⋆	3	3⋆	3⋆	3	4⋆
3	4	4	10	9	3
7⋆	9⋆	8⋆	8⋆	1⋆	7⋆
7	9	8	8	1	7
6	7⋆	4⋆	9	7⋆	4⋆
3	4	3	5	3	2
3	5	2	3	2	1
1	0	1	0	0	0
0	7⋆	6⋆	0	6⋆	6⋆
1	7	5	2	7	5
4	5⋆	1⋆	4	4⋆	1⋆
0	4	3	0	1	1
1	7⋆	5⋆	1	8⋆	7⋆
0	1	1	0	3	3

TAB. A.19 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

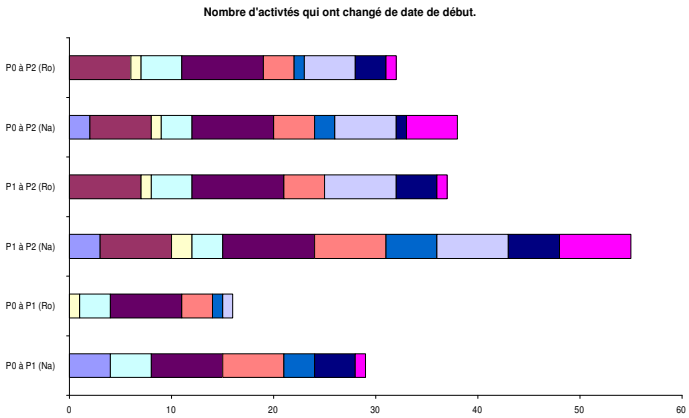


FIG. A.19 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

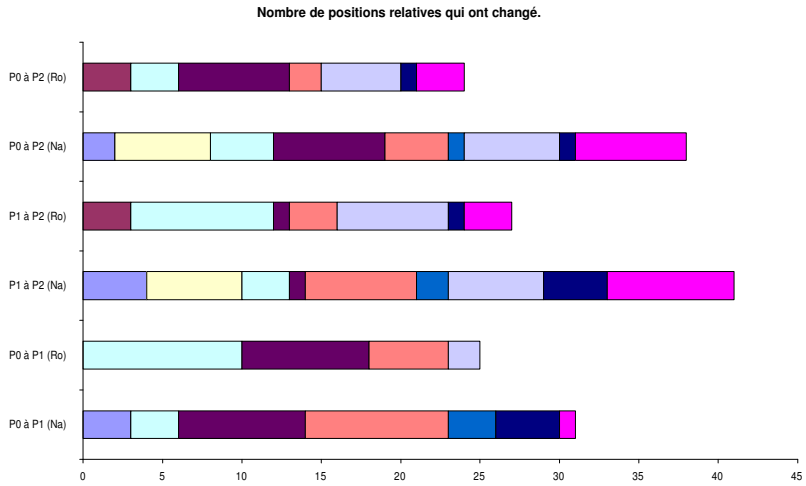


FIG. A.20 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

SDif.			**MDif.**		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
21	14	7	9	9	4
0	0	0	0	0	0
0	14⋆	12⋆	0	2⋆	2⋆
0	21	19	0	9	9
0	14⋆	13⋆	0	13⋆	13⋆
2	1	2	2	1	2
8⋆	7	9⋆	5⋆	5	4⋆
25	25	8	12	13	5
51⋆	26⋆	51⋆	11⋆	7⋆	11⋆
51	26	51	11	7	11
28	38⋆	19⋆	9	16⋆	7⋆
13	19	15	6	7	7
30	39	9	10	10	6
1	0	1	1	0	1
0	53⋆	43⋆	0	26⋆	26⋆
7	53	36	7	26	26
19	30⋆	8⋆	9	11⋆	8⋆
0	23	20	0	9	9
2	22⋆	19⋆	2	15⋆	15⋆
0	5	5	0	5	5

TAB. A.20 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

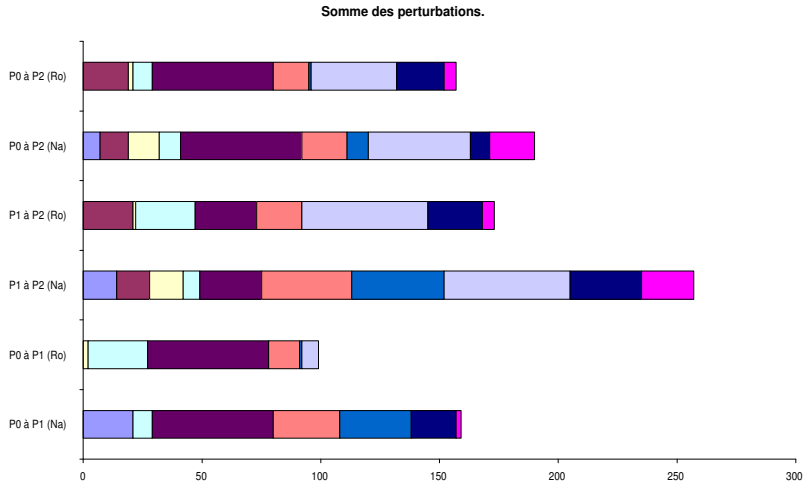


FIG. A.21 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

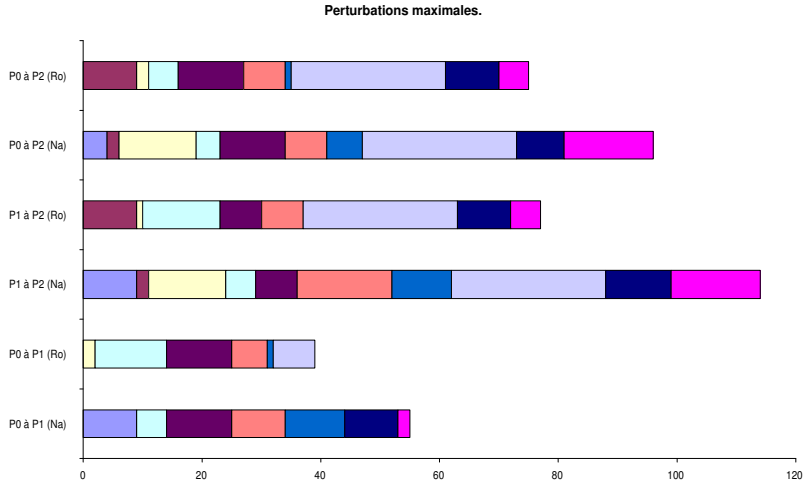


FIG. A.22 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série d'activités
Problèmes comportant 22 activités et 4 ressources

Diff.				Posi.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
7	3	14⋆	12⋆	9	4	26⋆	25⋆
8	3	13	14	31	1	7	33
6	0	14⋆	14⋆	6	0	29⋆	20⋆
6	13	8	12	25	18	23	10
15⋆	8	5	15⋆	28⋆	13	4	27⋆
16	11	8	16	30	15	9	25
7	9	18⋆	14⋆	14	8	30⋆	18⋆
8	2	7	10	4	2	8	9
10	4	8	11	24	5	18	28
3	6	8	11	12	12	25	22
7	14	7	13	10	28	15	34
13	2	10	12	31	4	26	16
9	12⋆	2	12⋆	36	17⋆	3	40⋆
7	12	1	13	34	21	1	38
7	10⋆	8	11⋆	33	11⋆	19	38⋆
7	8	6	11	29	26	15	29
2	0	3	4	5	0	1	4
4	3	2	4	4	1	1	6
7⋆	9	2	6⋆	19⋆	12	4	28⋆
12	7	10	15	25	12	15	32

TAB. A.21 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

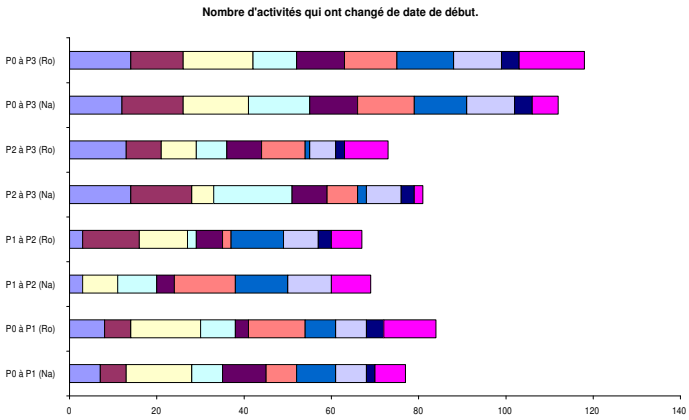


FIG. A.23 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

FIG. A.24 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

SDif.				MDif.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
25	7	55⋆	66⋆	7	3	24⋆	23⋆
68	6	22	77	24	2	4	24
13	0	47⋆	46⋆	4	0	23⋆	25⋆
41	38	33	31	17	13	14	9
79⋆	19	7	77⋆	18⋆	9	3	19⋆
78	26	14	74	19	9	7	13
18	18	94⋆	59⋆	5	6	11⋆	10⋆
12	3	10	18	3	2	3	6
60	7	25	74	14	3	9	13
24	18	49	55	16	7	14	14
12	59	23	71	3	13	12	14
66	2	55	28	14	1	14	9
59	47⋆	3	63⋆	21	9⋆	2	14⋆
52	75	2	101	13	13	2	13
47	32⋆	26	64⋆	10	8⋆	10	10⋆
33	75	27	69	9	17	14	12
16	0	5	15	10	0	3	7
9	6	6	19	3	4	4	7
32⋆	32	2	46⋆	12⋆	14	1	14⋆
55	30	23	78	13	16	9	19

TAB. A.22 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

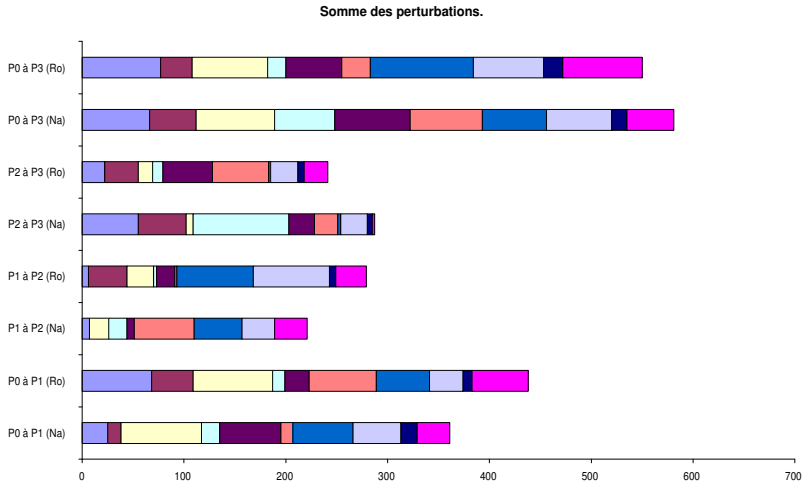


FIG. A.25 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

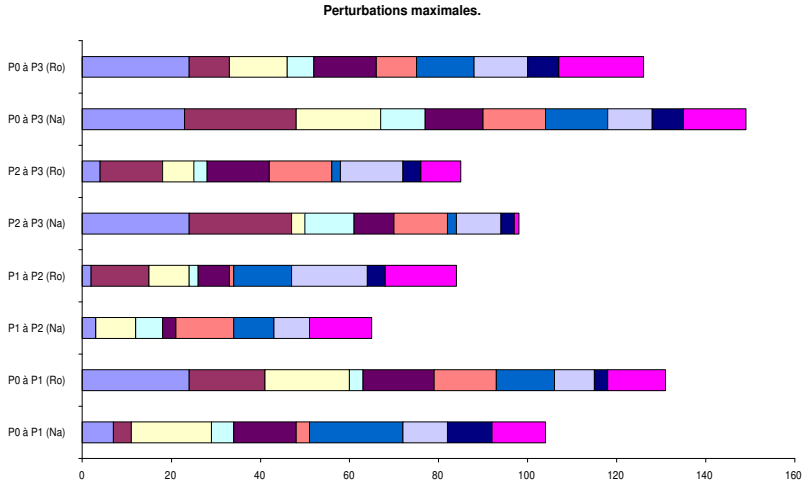


FIG. A.26 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série d'activités
Problèmes comportant 32 activités et 1 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
5	8	4	23⋆	20⋆	5	5	7	71⋆	51⋆
0	9	8	17	14	0	14	5	40	30
0	0	2	0	2	0	0	4	0	4
4	6	4	0	1	5	11	11	0	3
0	1	0	0	1	0	6	0	0	6
1	2	4	7	1	0	6	6	15	10
0	0	0	2	2	0	0	0	6	5
10	0	0	0	10	20	0	0	0	20
7	0	0	2	7	14	0	0	2	14
10	0	2	2	10	32	0	10	6	30
0	0	3	13	13	0	0	2	41	40
0	10	4	8	11	0	39	11	28	41
0	0	3	8	9	0	0	7	32	37
1	1	8	14	11	0	0	37	63	66
1	2	0	13	14	6	6	0	40	41
0	1	9	6	10	0	3	29	11	33
0	0	0	0	0	0	0	0	0	0
8	0	1	7	7	36	0	0	7	33
0	0	0	8	8	0	0	0	40	38
0	7	0	8	0	0	8	0	11	0

TAB. A.23 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et une ressource. Le cas où une série d'activités est ajoutée.*

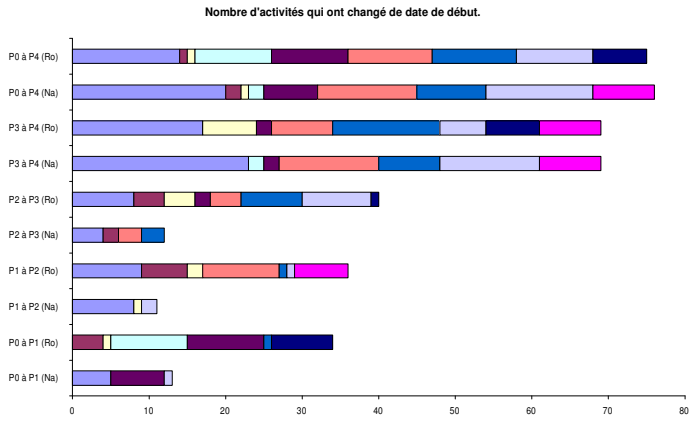


FIG. A.27 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est ajoutée.*

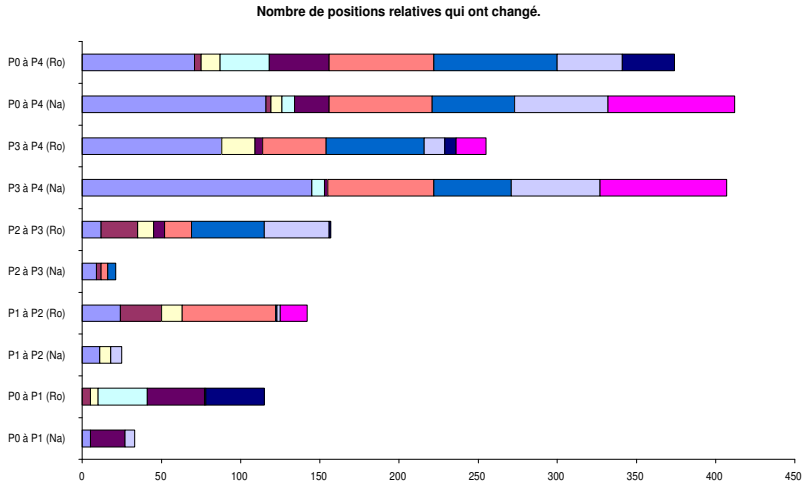


FIG. A.28 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est ajoutée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
5	11	9	145⋆	116⋆	1	4	3	17⋆	16⋆
0	24	12	88	71	0	8	4	16	13
0	0	3	0	3	0	0	2	0	2
5	26	23	0	4	2	18	18	0	4
0	7	0	0	7	0	7	0	0	7
5	13	10	21	12	5	8	4	4	12
0	0	0	8	8	0	0	0	4	4
31	0	0	0	31	8	0	0	0	8
22	0	0	2	22	13	0	0	1	13
36	0	7	5	38	13	0	4	3	13
0	0	4	67	65	0	0	2	20	20
0	59	17	40	66	0	17	8	10	17
0	0	5	49	52	0	0	3	15	15
1	1	46	62	78	1	1	12	10	22
6	7	0	56	59	6	5	0	15	15
0	2	41	13	41	0	2	9	8	8
0	0	0	0	0	0	0	0	0	0
37	0	1	7	33	12	0	1	1	12
0	0	0	80	80	0	0	0	21	21
0	17	0	19	0	0	9	0	9	0

TAB. A.24 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et une ressource. Le cas où une série d'activités est ajoutée.*

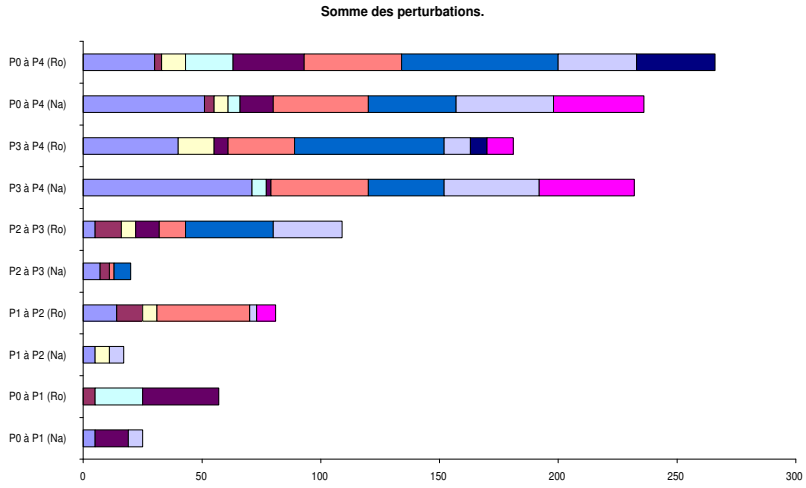


FIG. A.29 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est ajoutée.*

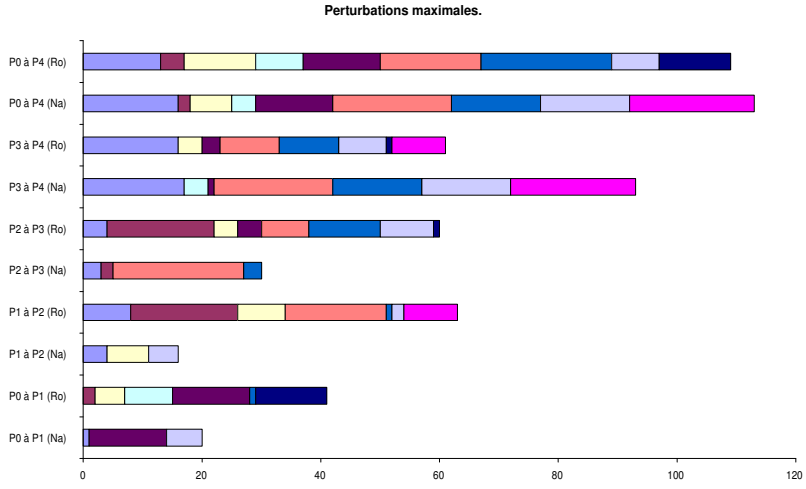


FIG. A.30 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série d'activités
Problèmes comportant 32 activités et 2 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
5	2	6	3	9	29	6	15	7	37
8	10	13	10	11	33	27	26	26	41
0	7	14	0	14	0	26	38	0	47
0	12	18	6	14	0	41	58	7	65
0	6	2	19⋆	18⋆	0	30	2	68⋆	59⋆
13	8	13	23	21	38	9	37	57	54
5	6	4	7	8	4	5	12	7	14
8	11	11	25	23	19	18	20	129	97
7	0	17⋆	7	14⋆	7	0	35⋆	16	25⋆
7	10	15	10	17	11	51	34	35	63
0	0	0	1	1	0	0	0	3	2
8	1	11	9	6	31	0	20	10	28
1	0	7	14	10	1	0	7	28	33
6	6	2	2	7	22	19	10	7	41
2	11	11	0	13	8	41	58	0	51
11	14	11	9	15	20	33	51	20	62
12	12	10	0	14	73	43	33	0	68
12	6	10	8	15	93	31	59	45	84
1	10	0	0	9	1	21	0	0	22
0	14	3	11	12	0	46	9	18	38

TAB. A.25 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et 2 ressources. Le cas où une série d'activités est ajoutée.*

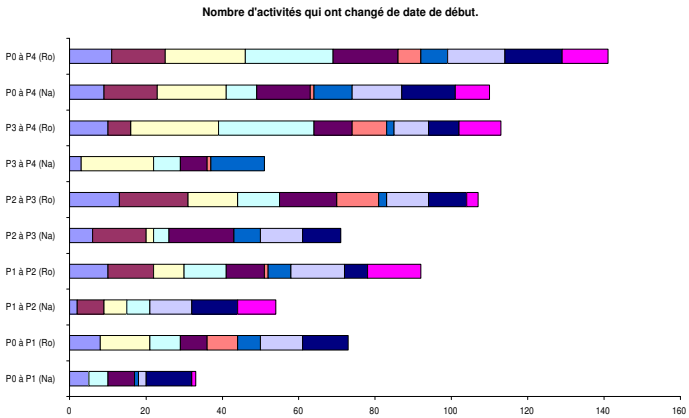


FIG. A.31 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est ajoutée.*

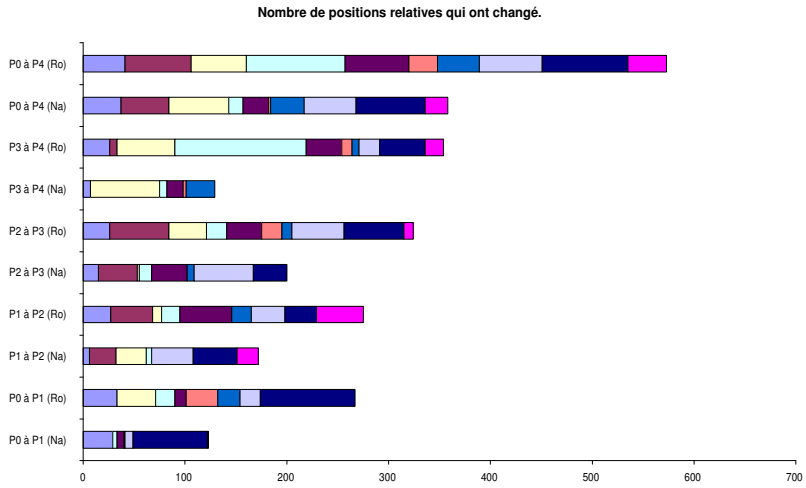


FIG. A.32 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est ajoutée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
21	4	19	11	35	9	3	8	5	11
33	33	38	26	45	10	7	7	6	13
0	26	46	0	58	0	13	10	0	13
0	36	64	15	67	0	10	13	4	11
0	23	2	100⋆	94⋆	0	8	1	17⋆	17⋆
48	9	47	87	86	11	2	11	9	12
6	10	12	11	17	2	3	5	3	6
30	28	22	303	269	8	8	3	29	29
18	0	87⋆	25	67⋆	4	0	40⋆	18	22⋆
21	94	72	52	142	8	23	14	24	23
0	0	0	4	4	0	0	0	4	4
71	3	46	16	62	15	3	20	5	15
3	0	10	42	43	3	0	3	12	12
34	43	19	16	72	15	21	16	12	21
10	47	62	0	75	5	7	23	0	23
16	45	70	16	91	5	7	26	6	23
113	51	41	0	113	40	18	25	0	25
142	36	66	45	131	40	18	20	26	25
1	35	0	0	34	1	16	0	0	16
0	78	14	25	59	0	20	8	6	19

TAB. A.26 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et 2 ressources. Le cas où une série d'activités est ajoutée.*

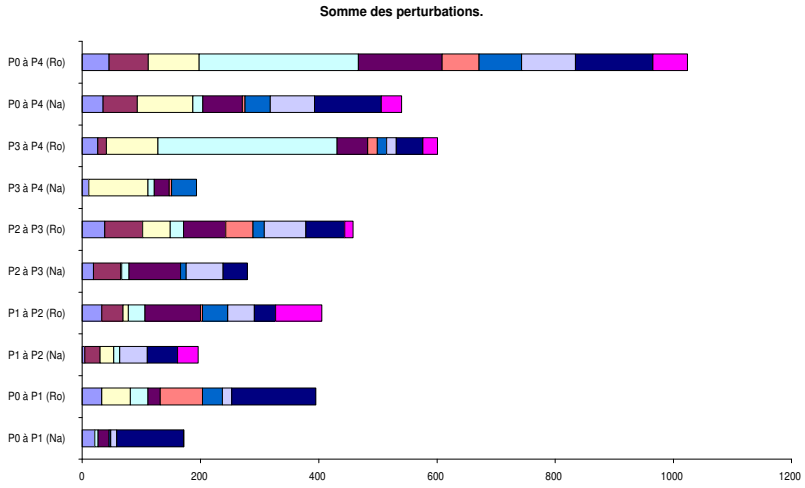


FIG. A.33 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est ajoutée.*

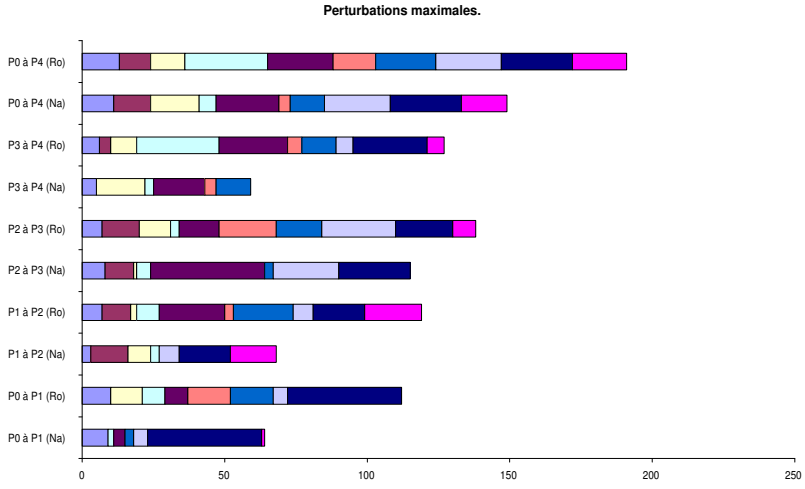


FIG. A.34 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série d'activités
Problèmes comportant 32 activités et 3 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
19⋆	0	0	6	17⋆	4⋆	0	0	17	21⋆
21	7	9	9	19	52	12	26	25	44
3	0	1	5	7	20	0	8	3	17
10	8	12	12	12	55	28	47	46	53
0	20⋆	4	11	21⋆	0	69⋆	4	12	68⋆
4	22	9	0	21	2	65	23	0	65
14	0	0	4	14	77	0	0	16	76
17	9	4	9	17	83	56	21	14	85
0	1	0	0	1	0	1	0	0	1
5	2	3	12	12	7	7	10	27	24
0	4	2	3	7	0	3	3	9	12
8	8	0	8	11	14	25	0	23	32
1	8	5	13	12	3	37	18	29	42
2	8	4	6	11	5	19	0	20	34
3	10	0	0	12	5	33	0	0	36
1	12	1	11	7	4	34	2	9	34
0	0	0	16	14	0	0	0	59	50
4	9	13	12	10	8	16	32	45	50
0	0	2	14	12	0	0	8	21	22
10	1	4	12	10	9	2	14	40	22

TAB. A.27 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et 3 ressources. Le cas où une série d'activités est ajoutée.*

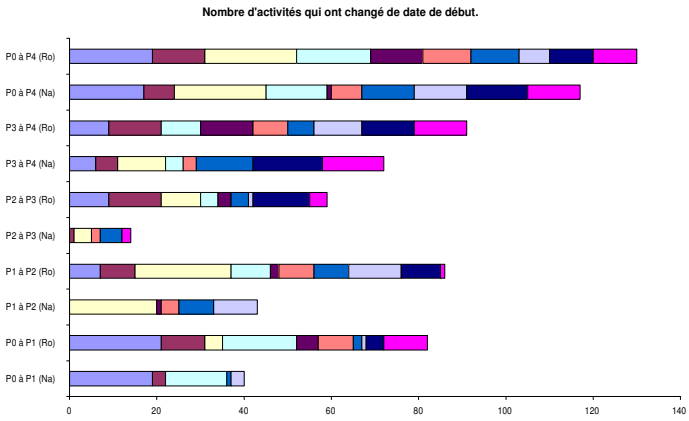


FIG. A.35 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est ajoutée.*

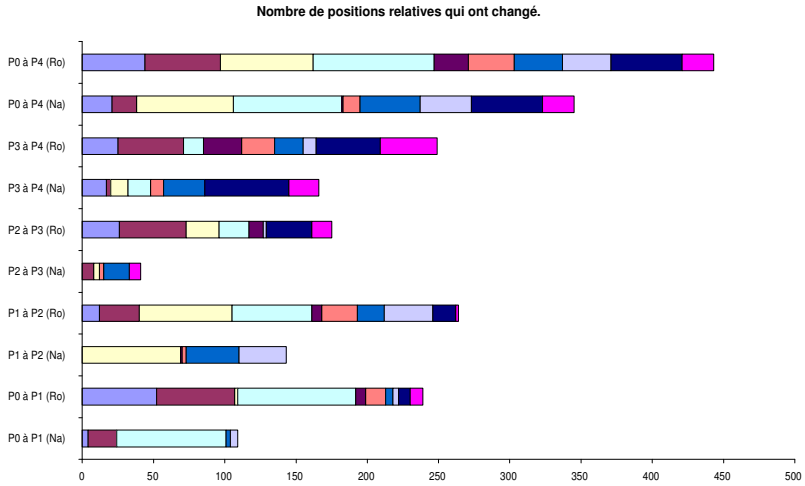


FIG. A.36 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est ajoutée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
43⋆	0	0	21	48⋆	8⋆	0	0	8	10⋆
88	14	32	31	78	9	5	9	15	12
44	0	12	8	40	27	0	12	4	15
94	42	74	71	90	19	19	15	16	19
0	140⋆	10	23	146⋆	0	25⋆	4	4	25⋆
4	131	36	0	151	1	25	12	0	25
148	0	0	26	155	17	0	0	19	17
163	88	26	22	209	20	17	17	8	21
0	1	0	0	1	0	1	0	0	1
8	8	12	43	40	2	7	9	9	8
0	6	3	12	17	0	2	2	8	8
20	57	0	38	55	6	10	0	16	9
9	50	27	48	68	9	25	13	9	12
12	26	4	30	52	9	7	1	9	9
8	40	0	0	46	6	9	0	0	9
7	43	1	12	40	7	9	1	2	9
0	0	0	88	86	0	0	0	18	18
9	23	46	66	85	4	7	9	24	24
0	0	9	36	32	0	0	7	5	6
12	1	18	64	31	3	1	8	24	7

TAB. A.28 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et 3 ressources. Le cas où une série d'activités est ajoutée.*

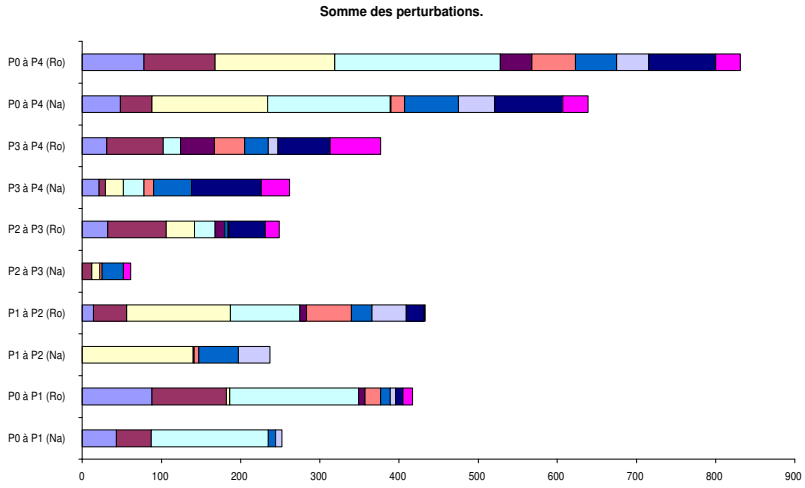


FIG. A.37 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est ajoutée.*

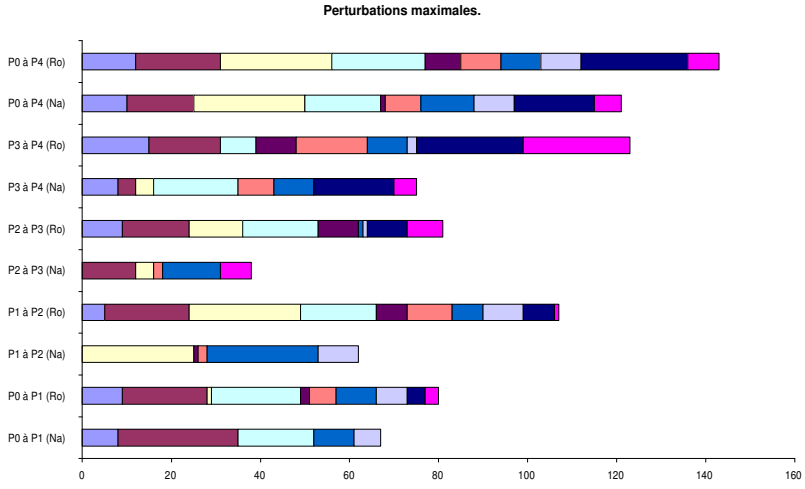


FIG. A.38 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série d'activités
Problèmes comportant 32 activités et 4 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
0	0	0	3	3	0	0	0	7	6
1	10	11	2	12	0	41	51	3	36
0	4	0	0	4	0	3	0	0	3
2	3	1	4	5	2	3	1	2	4
0	4	1	0	3	0	17	13	0	5
4	3	0	3	4	3	3	0	5	6
4	4	9	0	4	9	18	30	0	11
1	8	4	5	3	5	19	17	20	20
0	0	6	0	6	0	0	16	0	14
0	0	4	5	6	0	0	17	20	18
0	0	1	4	5	0	0	3	7	10
0	0	3	3	4	0	0	12	14	8
0	2	0	11	11	0	10	0	54	53
3	7	4	19	15	12	41	10	73	78
0	0	12	1	12	0	0	59	2	61
0	0	14	4	12	0	0	72	14	65
0	0	0	0	0	0	0	0	0	0
1	4	2	2	4	4	10	2	2	6
4	1	3	12	13	34	16	18	27	50
7	13	9	13	15	42	52	29	40	52

TAB. A.29 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

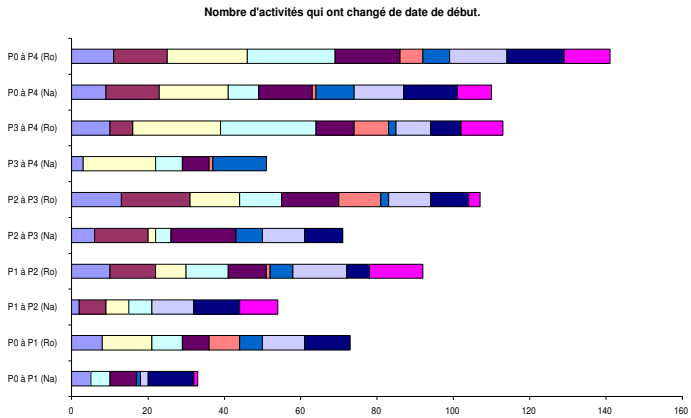


FIG. A.39 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

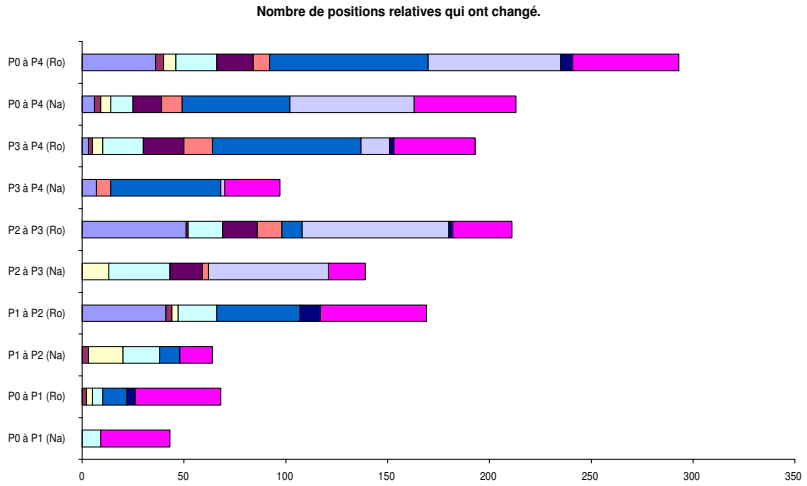


FIG. A.40 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
0	0	0	10	10	0	0	0	8	8
1	61	71	13	56	1	16	17	10	11
0	5	0	0	5	0	2	0	0	2
2	4	1	5	6	1	2	1	2	2
0	19	10	0	9	0	10	10	0	6
4	4	0	8	10	1	2	0	6	6
11	18	34	0	14	8	8	11	0	8
8	23	16	19	25	8	9	7	15	9
0	0	13	0	13	0	0	7	0	7
0	0	16	19	17	0	0	7	15	7
0	0	3	12	15	0	0	3	5	5
0	0	15	23	14	0	0	11	15	5
0	12	0	60	70	0	11	0	15	15
12	56	11	96	110	8	32	6	13	26
0	0	160	4	164	0	0	21	4	21
0	0	136	16	134	0	0	23	8	20
0	0	0	0	0	0	0	0	0	0
6	11	2	2	5	6	7	1	1	2
48	22	22	27	58	28	22	14	6	15
54	76	48	56	69	28	22	15	27	15

TAB. A.30 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et 4 ressources. Le cas où une série d'activités est ajoutée.*

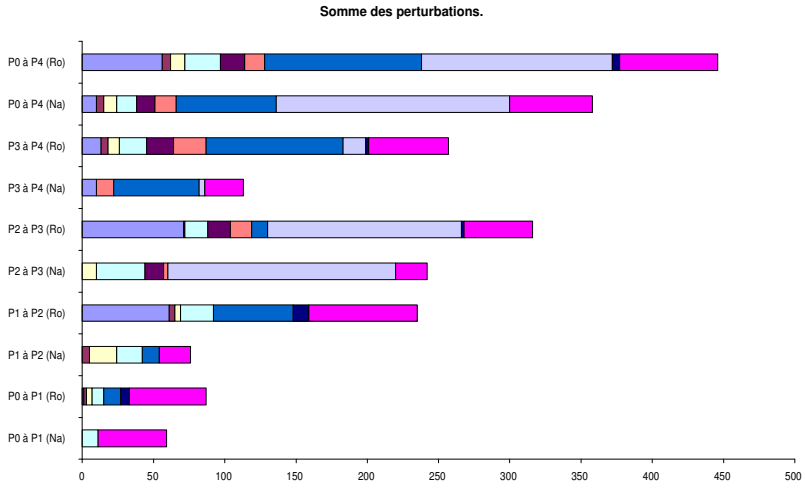


FIG. A.41 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

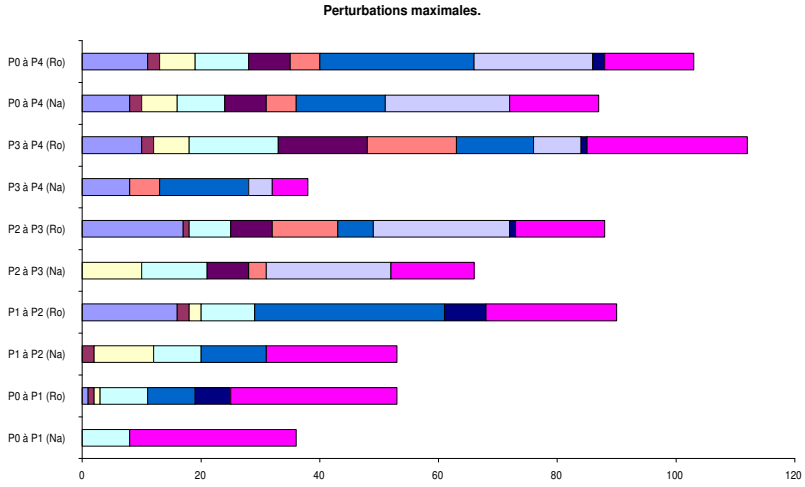


FIG. A.42 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est ajoutée.*

Résultats obtenus pour la stabilté
Retrait d'une série d'activités
Problèmes comportant 12 activités et 4 ressources

Diff.			Posi.		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
4	4	1	4	5	2
0	0	0	0	0	0
2	1⋆	2⋆	1	1⋆	0⋆
3	2	2	4	2	5
4⋆	3	3⋆	6⋆	4	5⋆
3	7	8	1	7	8
6	4	3	4	4	1
6	4	2	5	4	1
3	3⋆	5⋆	4	4⋆	1⋆
3	5	5	4	6	3
6⋆	4	4⋆	7⋆	10	3⋆
4	2	4	6	5	1
9⋆	4	7⋆	3⋆	4	1⋆
9	1	8	2	1	2
1	5⋆	5⋆	1	8⋆	7⋆
1	7	7	1	12	11
6⋆	1	5⋆	9⋆	6	3⋆
6	2	5	7	9	3
2	5⋆	6⋆	8	9⋆	4⋆
1	6	6	7	10	4

TAB. A.31 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série d'activités est retirée.*

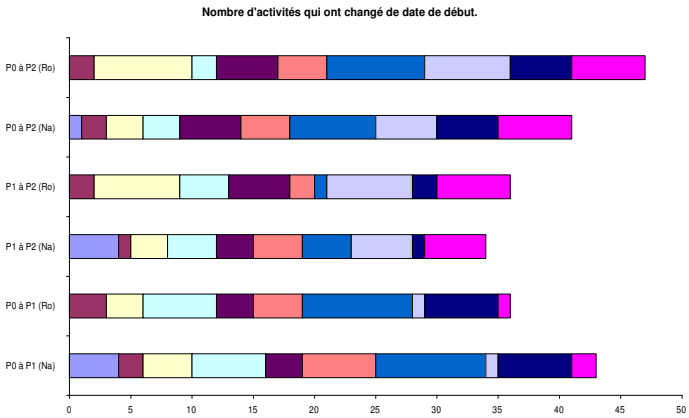


FIG. A.43 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est retirée.*

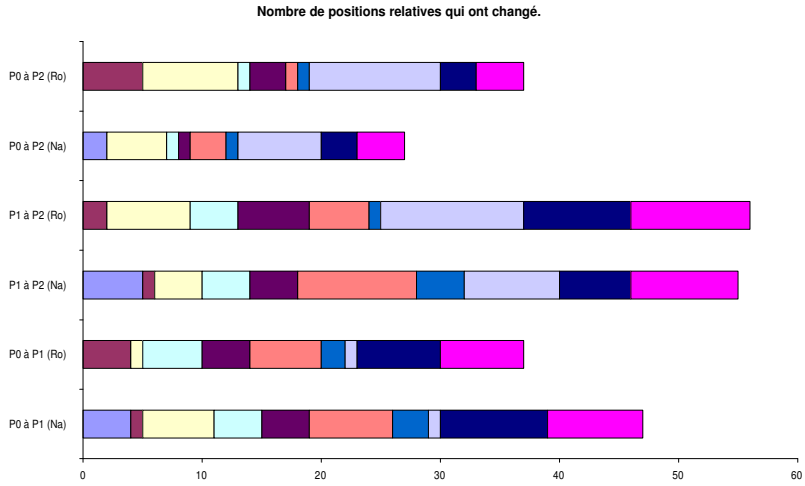


FIG. A.44 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est retirée.*

SDif.			**MDif.**		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
26	22	4	9	9	4
0	0	0	0	0	0
7	1⋆	6⋆	4	1⋆	3⋆
20	5	23	16	4	20
20⋆	3	15⋆	6⋆	1	7⋆
13	39	48	6	7	7
16	11	7	5	5	3
12	8	4	5	5	3
14	15⋆	11⋆	10	9⋆	5⋆
14	21	13	10	9	7
43⋆	20	30⋆	15⋆	7	8⋆
42	18	25	21	14	7
61⋆	27	45⋆	16⋆	16	9⋆
43	3	40	6	3	6
6	53 ⋆	59 ⋆	6	15⋆	15⋆
6	78	84	6	15	15
64⋆	14	42⋆	22⋆	14	10⋆
56	22	42	22	14	10
17	30⋆	25⋆	15	11⋆	10⋆
15	32	25	15	11	10

TAB. A.32 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série d'activités est retirée.*

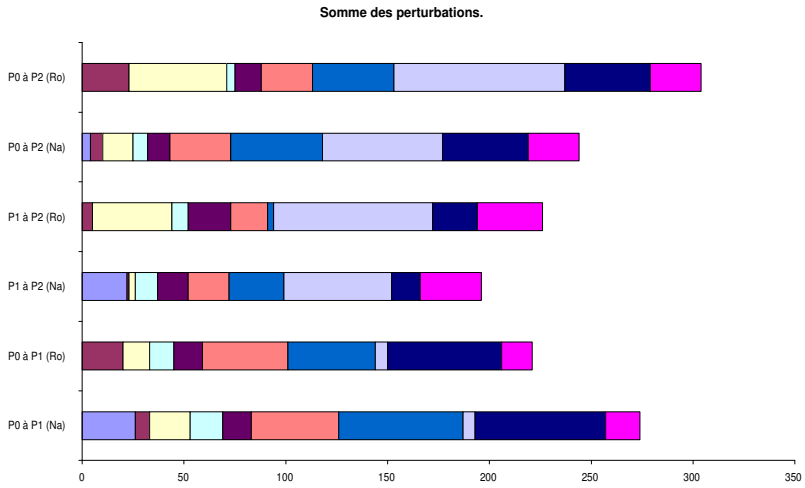


FIG. A.45 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est retirée.*

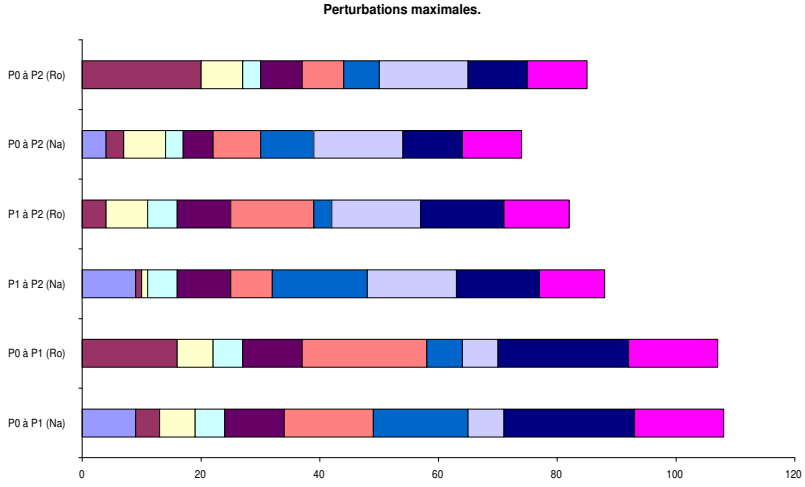


FIG. A.46 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série d'activités est retirée.*

Résultats obtenus pour la stabilté
Retrait d'une série d'activités
Problèmes comportant 22 activités et 4 ressources

Diff.				Posi.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
13	15	7	11	55	52	36	37
18	15	6	14	75	52	32	48
16⋆	6	13	11⋆	21⋆	18	16	16⋆
16	3	5	11	21	13	8	20
14⋆	1	2	12⋆	21⋆	6	6	11⋆
15	9	7	14	25	35	11	26
19⋆	6	4	17⋆	25⋆	16	21	3⋆
19	5	4	17	24	13	10	12
8	7	13⋆	12⋆	26	12	32⋆	38⋆
9	6	14	13	22	23	41	32
6	13⋆	5	12⋆	13	22⋆	5	19⋆
10	12	7	14	28	28	8	36
10⋆	7	1	5⋆	27⋆	16	2	14⋆
9	9	3	6	28	15	5	15
6	4	11⋆	10⋆	10	11	23⋆	26⋆
11	8	9	11	41	12	14	35
6	2	4⋆	4⋆	8	4	3⋆	9⋆
6	4	5	3	2	6	6	9
13⋆	5	3	12⋆	28⋆	10	5	19⋆
14	9	1	13	16	27	3	23

TAB. A.33 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série d'activités est retirée.*

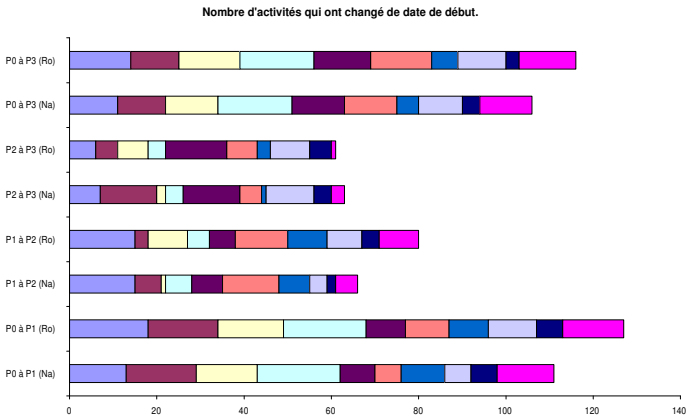


FIG. A.47 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est retirée.*

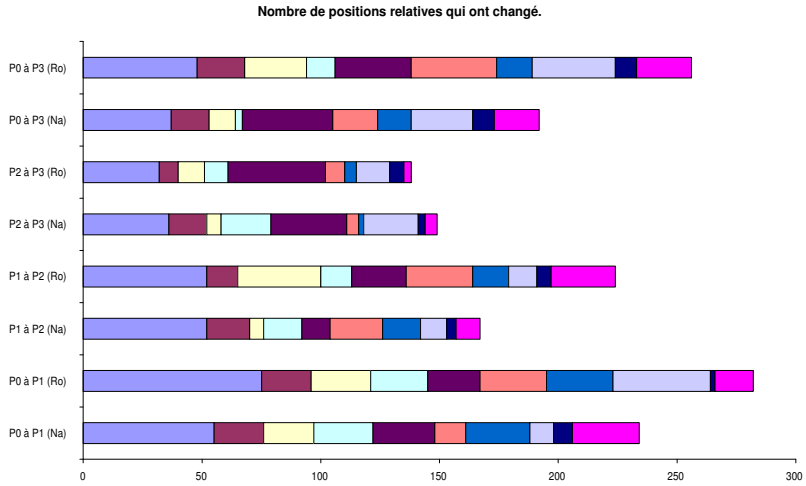


FIG. A.48 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est retirée.*

SDif.				MDif.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
89	97	58	67	17	20	22	16
115	97	57	108	24	13	33	14
45⋆	29	39	36⋆	10⋆	10	10	11⋆
47	20	14	46	10	10	5	14
45⋆	7	13	33⋆	7⋆	7	7	7⋆
46	53	23	60	8	13	8	12
177⋆	28	38	146⋆	23⋆	19	19	12⋆
177	19	18	142	14	8	8	14
38	19	87⋆	100⋆	8	5	18⋆	17⋆
33	39	103	99	8	19	18	17
20	54⋆	11	48⋆	5	24⋆	7	25⋆
50	55	22	80	26	20	8	26
45⋆	24	7	24⋆	12⋆	10	7	9⋆
53	27	9	26	17	11	6	9
16	18	53⋆	59⋆	8	8	16⋆	17⋆
76	24	32	77	20	8	11	12
15	10	12⋆	18⋆	6	6	5⋆	6⋆
6	11	18	20	1	6	10	14
51⋆	17	14	34⋆	13⋆	5	12	10⋆
37	52	7	54	9	24	7	24

TAB. A.34 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série d'activités est retirée.*

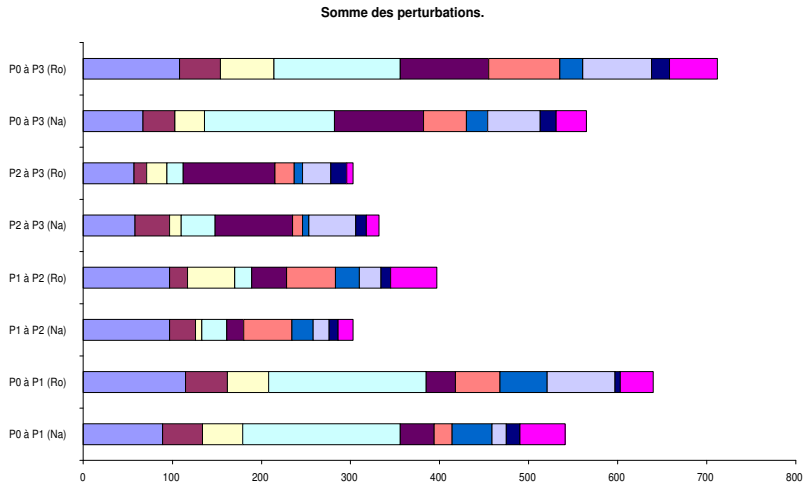


FIG. A.49 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est retirée.*

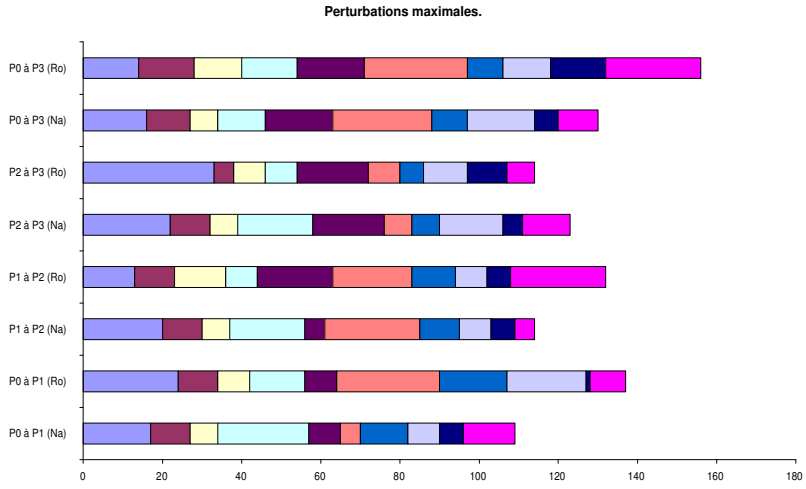


FIG. A.50 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série d'activités est retirée.*

Résultats obtenus pour la stabilté
Retrait d'une série d'activités
Problèmes comportant 32 activités et une ressource

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
27⋆	20	3	2	26⋆	84⋆	42	19	9	63⋆
27	4	3	9	23	70	37	21	37	61
16	12	13	14⋆	16⋆	55	34	62	51⋆	73⋆
12	11	12	5	5	39	18	30	21	5
6	5	5	8	12	26	8	11	14	24
13	7	10	7	11	49	12	18	22	42
7	2	5	3	6	41	13	31	25	19
11	3	10	3	4	60	17	54	28	15
8	5	9	3	10	49	17	26	10	34
8	9	15	4	13	49	27	47	6	51
8	9	7	14⋆	16⋆	35	53	42	38⋆	24⋆
17	4	10	14	19	65	34	42	43	54
5	16⋆	7	9	15⋆	34	37⋆	20	27	30⋆
9	18	9	4	17	31	62	45	63	78
9	5	3	2	8	31	22	21	11	12
10	8	11	3	14	35	21	43	5	30
19	6	8	9⋆	21⋆	37	28	38	19⋆	11⋆
19	9	3	11	22	68	49	25	52	50
5	8	5	2	7	48	40	16	8	5
5	8	2	12	13	48	40	14	26	21

TAB. A.35 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et une ressource. Le cas où une série d'activités est retirée.*

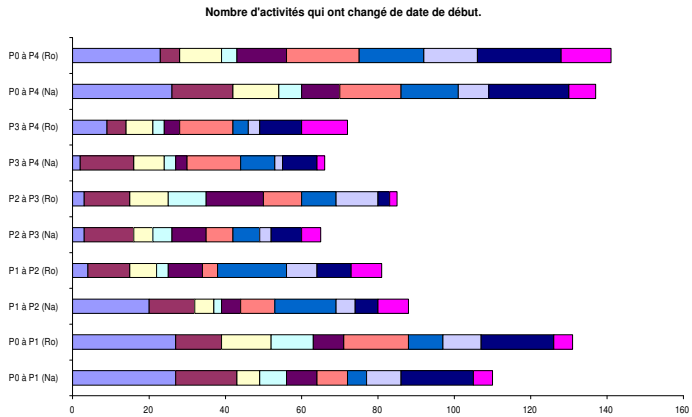


FIG. A.51 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est retirée.*

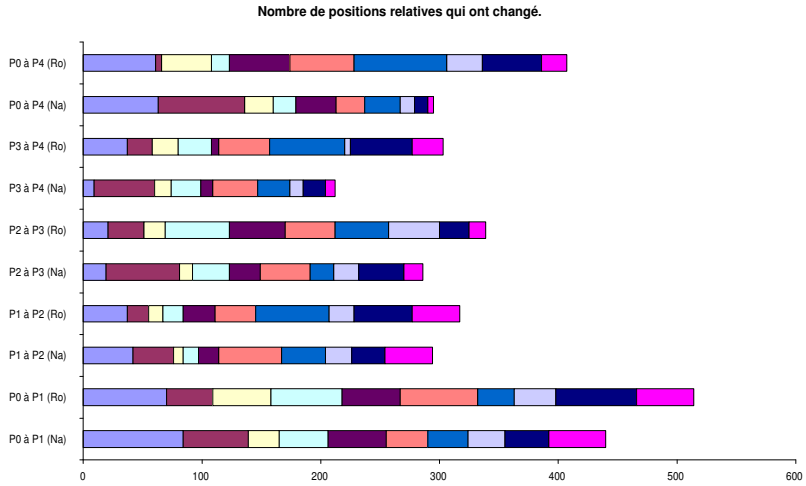


FIG. A.52 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est retirée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
220⋆	59	20	8	211⋆	20⋆	16	9	7	17⋆
251	43	24	50	237	20	14	11	24	37
113	62	129	111⋆	169⋆	22	13	29	22⋆	26⋆
69	31	59	42	12	23	7	26	31	6
36	8	27	63	81	10	3	17	15	15
65	17	36	59	89	15	10	10	15	15
57	18	51	28	34	17	9	19	10	19
87	23	83	32	31	17	11	22	11	21
77	21	41	25	64	13	7	7	15	12
77	46	77	8	84	13	19	25	3	14
35	51	53	55⋆	39⋆	13	20	21	17⋆	7⋆
76	30	49	55	80	21	12	17	15	21
22	47⋆	19	37	42⋆	6	15⋆	11	14	10⋆
25	75	45	56	104	21	17	20	17	4
34	19	24	12	19	9	7	9	6	9
36	24	61	6	40	9	9	11	3	8
46	25	37	32⋆	52⋆	9	15	15	8⋆	5⋆
68	54	22	72	89	12	15	15	14	16
79	70	26	14	13	37	37	14	7	5
79	70	23	55	47	37	37	14	13	15

TAB. A.36 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et une ressource. Le cas où une série d'activités est retirée.*

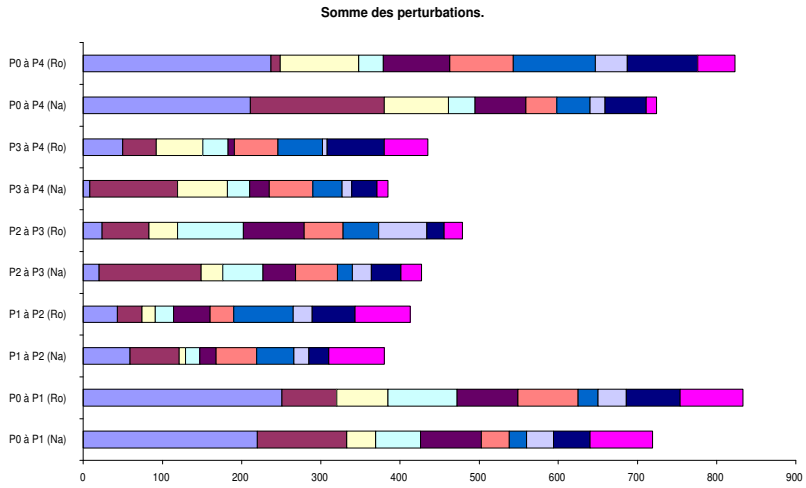


FIG. A.53 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est retirée.*

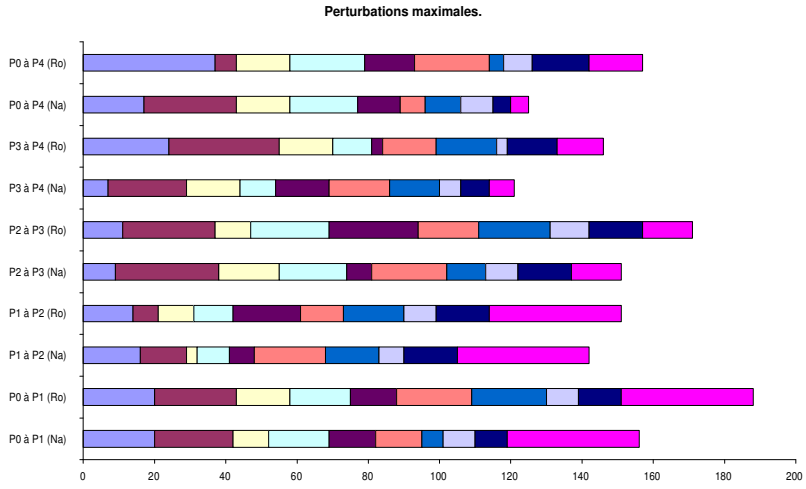


FIG. A.54 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et une ressource dans le cas où une série d'activités est retirée.*

Résultats obtenus pour la stabilté
Retrait d'une série d'activités
Problèmes comportant 32 activités et 2 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
5	3	8	6	8	39	9	29	17	20
17	11	13	11	15	107	47	45	35	77
5	9	8	3	6	52	31	35	23	17
13	9	10	12	11	98	46	38	56	47
4	7	5	4	6	16	46	28	15	35
9	4	16	9	12	35	19	57	31	58
16	14	9	11	13	59	51	28	34	32
13	9	7	12	13	26	59	27	70	54
26⋆	7	9	6	23⋆	63⋆	28	27	33	53⋆
24	11	10	8	22	109	34	27	26	86
7	8	12	6	8	20	24	40	17	34
4	6	8	3	6	21	16	26	7	11
3	6	3	5	6	34	30	34	39	29
7	10	6	12	12	52	52	51	63	57
3	4	2	16⋆	16⋆	34	32	20	78⋆	72⋆
11	12	8	15	15	66	63	27	75	63
14	11	18⋆	15⋆	16⋆	56	34	54⋆	33⋆	62⋆
9	3	11	12	13	73	6	24	46	72
9	6	4	3	6	50	34	13	5	26
14	11	3	15	16	71	47	10	74	93

TAB. A.37 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et 2 ressources. Le cas où une série d'activités est retirée.*

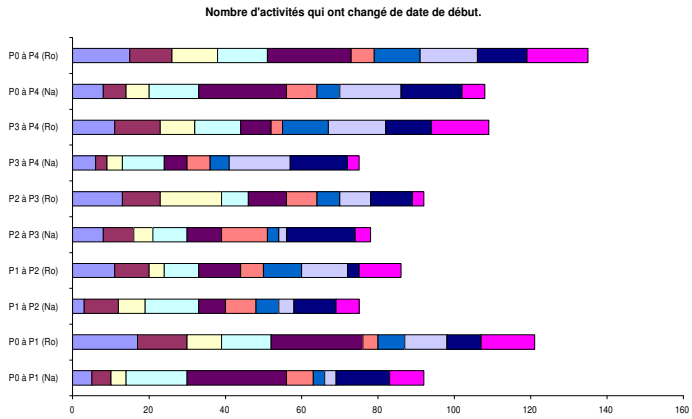


FIG. A.55 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est retirée.*

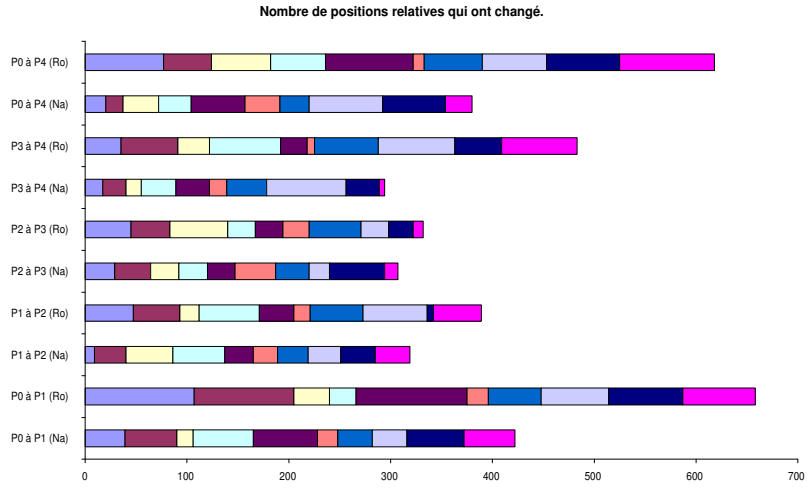


FIG. A.56 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est retirée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
39	18	44	26	29	10	8	10	5	5
142	59	50	54	117	28	13	10	14	28
48	42	40	30	22	15	11	15	12	12
112	53	39	70	63	22	11	15	21	15
11	37	23	13	38	4	19	9	6	19
33	14	79	17	57	19	9	11	4	9
75	70	39	41	33	12	14	11	11	5
32	58	24	76	62	13	13	9	20	10
103⋆	42	41	47	108⋆	19⋆	16	17	19	24⋆
231	50	54	52	217	41	13	16	16	41
23	35	58	29	49	8	16	12	10	14
20	27	36	9	18	8	8	8	5	7
25	25	27	32	23	12	8	12	12	7
39	58	53	73	66	12	15	19	19	16
23	24	13	87⋆	73⋆	8	9	9	16⋆	12⋆
61	69	31	90	68	16	20	21	20	16
66	43	75⋆	45⋆	114⋆	12	11	10⋆	7⋆	20⋆
106	5	35	75	134	24	2	22	22	24
67	51	21	12	43	18	19	10	7	18
114	76	27	130	159	19	20	15	24	24

TAB. A.38 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et 2 ressources. Le cas où une série d'activités est retirée.*

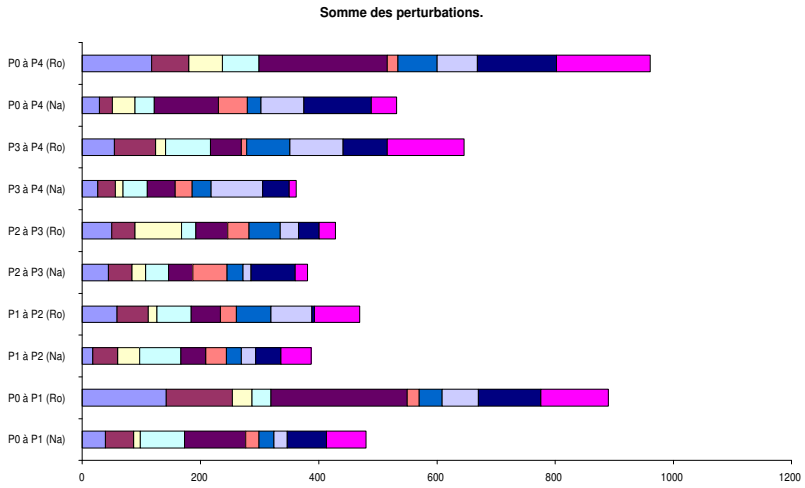


FIG. A.57 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est retirée.*

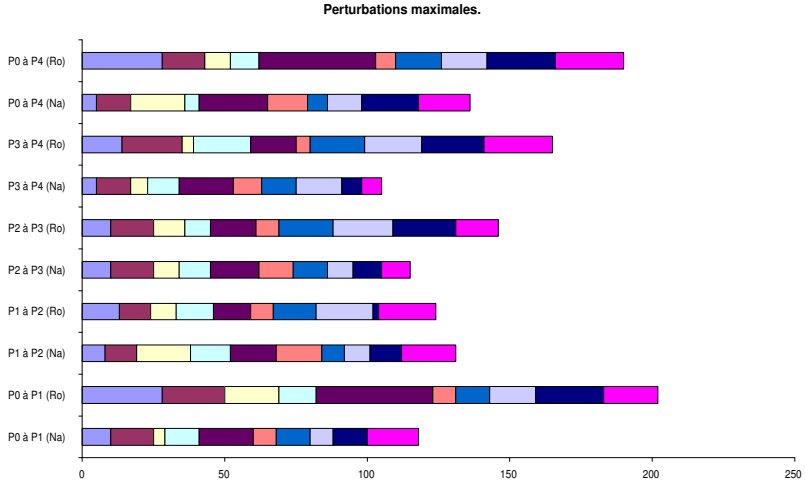


FIG. A.58 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et 2 ressources dans le cas où une série d'activités est retirée.*

Résultats obtenus pour la stabilté
Retrait d'une série d'activités
Problèmes comportant 32 activités et 3 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
18	6	3	3	15	82	26	21	11	59
13	5	5	5	7	55	36	23	14	27
6	4	9	4	8	59	40	34	25	10
16	4	10	3	16	97	54	39	30	44
3	2	5	4	3	22	18	14	18	6
11	12	6	4	9	36	41	20	18	26
4	13⋆	3	3	12⋆	8	48⋆	18	15	35⋆
7	14	10	7	9	12	46	35	28	24
11	11	8	4	11	63	33	20	12	31
16	9	5	3	11	48	29	16	7	20
1	3	2	1	1	6	19	11	4	5
12	13	4	2	2	48	59	16	7	7
14	7	12	12	11	56	44	35	43	40
16	4	6	9	12	67	33	31	74	59
8	19⋆	6	5	17⋆	71	57⋆	30	22	42⋆
7	22	5	13	23	74	69	31	68	73
4	10	6	1	9	46	21	18	3	19
13	13	5	16	14	64	69	14	52	67
13	7	13⋆	4	16⋆	25	27	26⋆	20	19⋆
15	7	14	3	18	48	25	42	9	45

TAB. A.39 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et 3 ressources. Le cas où une série d'activités est retirée.*

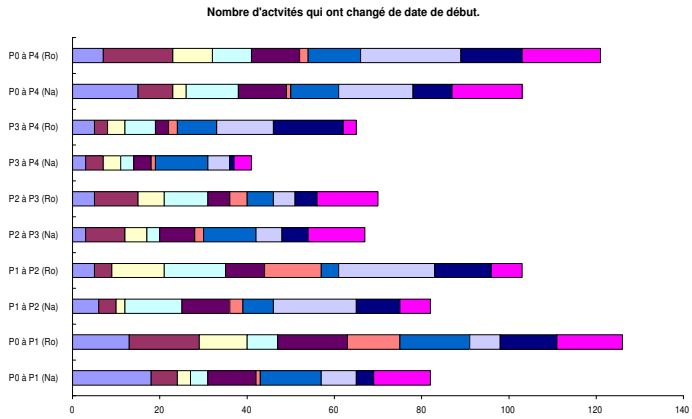


FIG. A.59 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est retirée.*

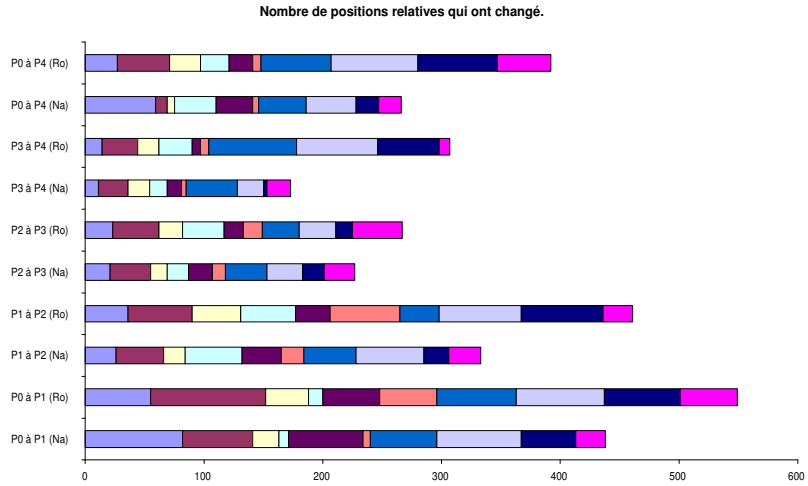


FIG. A.60 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est retirée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
129	40	22	14	101	18	19	19	7	15
82	45	32	18	48	15	16	16	8	19
76	57	57	30	17	21	24	24	15	4
137	63	60	35	80	22	23	23	17	18
17	16	12	15	6	8	11	6	7	4
42	47	20	22	50	13	13	12	12	23
8	132⋆	32	30	110⋆	2	39⋆	12	12	39⋆
15	115	75	43	73	5	19	20	11	13
79	62	37	27	61	22	22	16	10	16
59	32	16	7	20	16	7	9	9	22
8	21	11	6	8	8	8	6	6	8
85	97	14	8	9	28	28	6	6	8
73	43	47	54	48	11	16	13	15	11
109	37	37	92	83	25	13	11	27	17
94	112⋆	40	31	94⋆	25	22⋆	22	13	25⋆
94	135	42	95	151	25	22	21	21	25
60	39	19	7	32	25	8	13	7	8
101	110	17	70	135	25	22	10	16	25
57	77	91⋆	51	76⋆	18	21	21⋆	24	18⋆
84	72	93	11	101	18	19	19	7	18

TAB. A.40 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et 3 ressources. Le cas où une série d'activités est retirée.*

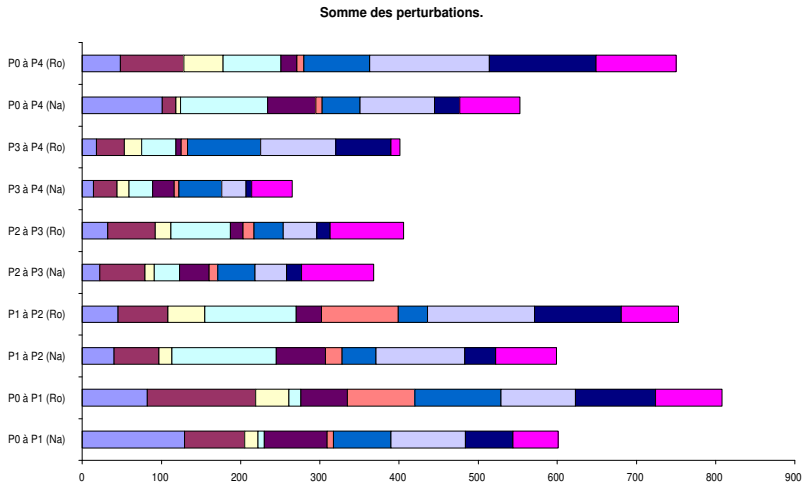


FIG. A.61 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est retirée.*

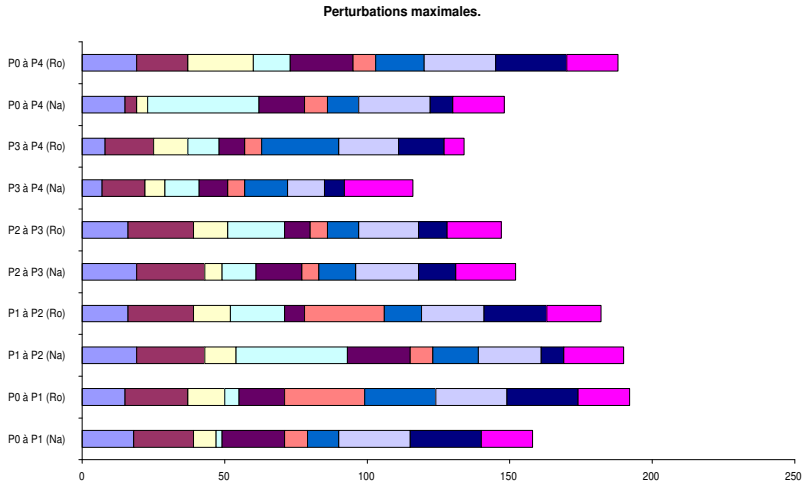


FIG. A.62 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et 3 ressources dans le cas où une série d'activités est retirée.*

Résultats obtenus pour la stabilté
Retrait d'une série d'activités
Problèmes comportant 32 activités et 4 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
7	3	21⋆	13	20⋆	38	23	92⋆	48	79⋆
15	8	22	10	20	60	21	94	65	95
6	6	3	3	6	15	19	10	9	9
11	5	3	3	7	44	25	13	20	16
6	12	2	11	11	43	39	20	40	40
12	12	2	2	3	56	30	23	19	9
13	11	1	3	10	25	44	13	24	29
3	4	2	6	5	0	37	18	30	8
7	4	10	4	8	40	25	37	25	34
14	9	7	8	12	55	64	47	55	55
5	9	5	15⋆	16⋆	54	40	21	41⋆	52⋆
6	12	10	17	18	58	41	41	70	57
6	11	5	2	8	61	58	25	26	5
5	4	6	5	8	57	53	24	33	14
11	16⋆	2	3	19⋆	58	45⋆	19	22	31⋆
12	22	10	7	20	69	40	27	21	38
6	7	5	9	5	37	52	5	21	11
5	10	9	2	8	36	61	23	8	42
14	8	13	13	16	51	14	30	34	39
16	13	11	5	14	50	36	31	37	28

TAB. A.41 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités et 4 ressources. Le cas où une série d'activités est retirée.*

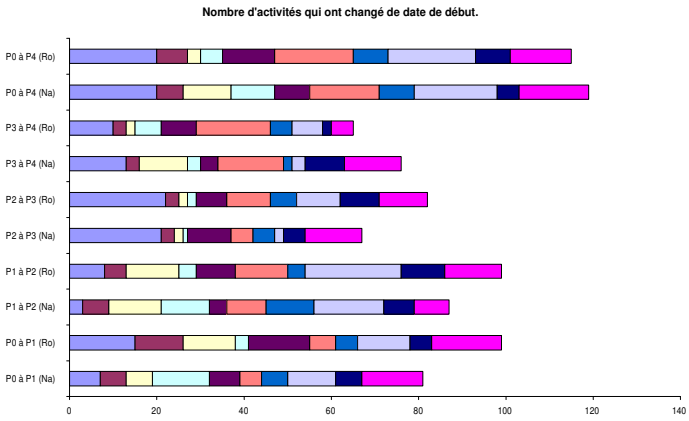


FIG. A.63 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est retirée.*

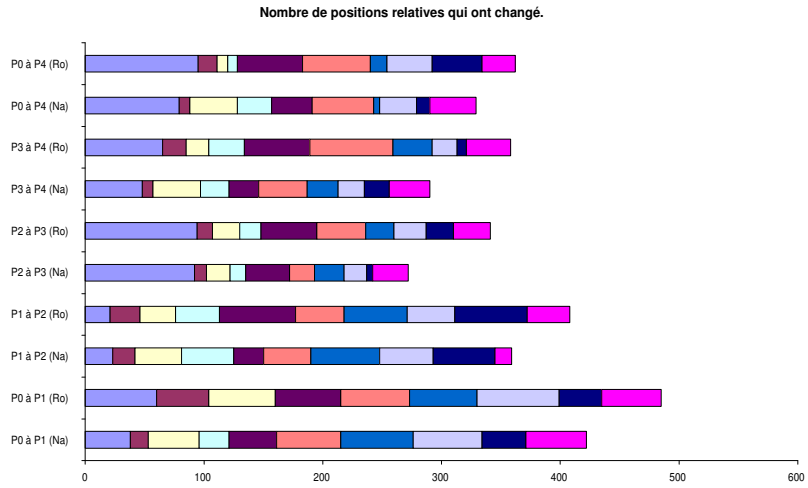


FIG. A.64 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est retirée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
50	26	254⋆	45	227⋆	18	18	30⋆	20	21⋆
77	31	236	66	222	18	16	32	21	20
20	20	10	9	16	6	10	6	6	6
54	21	10	22	25	13	11	8	11	13
47	48	24	64	54	18	7	14	14	17
61	42	25	22	9	18	8	14	14	4
21	36	7	20	25	4	12	7	11	4
3	34	13	34	10	1	15	10	15	5
38	24	49	28	47	13	8	14	14	13
59	66	68	73	75	13	28	29	29	22
51	46	28	90⋆	102⋆	14	11	12	10⋆	13⋆
58	48	63	113	105	14	11	17	18	17
53	57	33	21	9	12	18	18	11	2
52	50	35	30	18	12	18	18	11	8
81	60⋆	18	21	68⋆	21	14⋆	11	12	14⋆
82	68	82	68	36	13	15	8	9	13
47	84	10	43	15	30	30	2	21	9
46	123	70	20	124	30	42	28	19	42
64	13	43	50	63	27	3	19	21	29
70	43	38	54	49	15	10	10	29	19

TAB. A.42 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités et 4 ressources. Le cas où une série d'activités est retirée.*

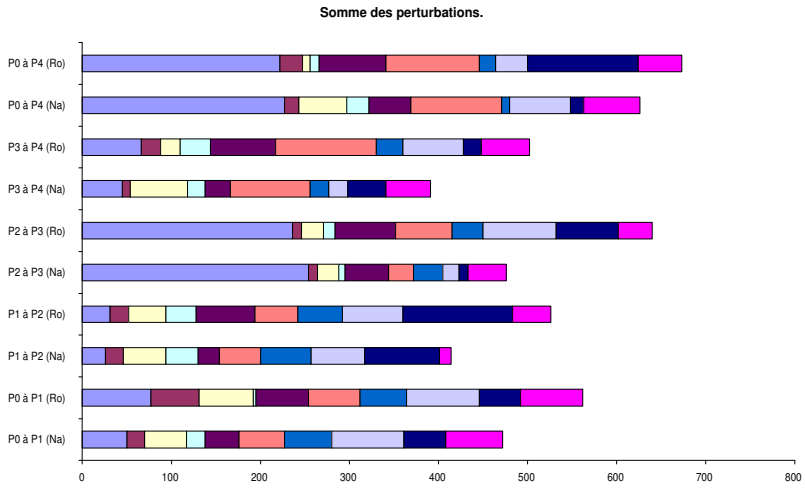


FIG. A.65 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est retirée.*

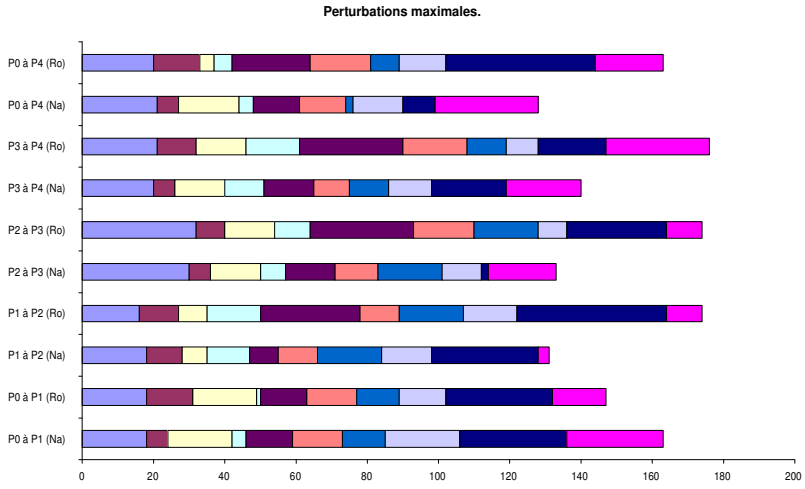


FIG. A.66 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités et 4 ressources dans le cas où une série d'activités est retirée.*

Résultats obtenus pour la stabilté
Ajout d'une série de contraintes de précédence
Problèmes comportant 12 activités et 4 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
3	7⋆	1	7⋆	5⋆	2	7⋆	1	7⋆	5⋆
4	7	1	7	5	6	5	5	3	5
5	2	3	8⋆	10⋆	11	9	11	6⋆	10⋆
4	4	0	8	10	5	12	0	9	9
9⋆	4	4⋆	4	9⋆	9⋆	4	4⋆	7	13⋆
7	6	4	4	9	10	5	4	7	13
3⋆	2	1⋆	2	4⋆	7⋆	2	1⋆	3	10⋆
3	2	3	0	4	9	3	4	0	10
6⋆	4⋆	8⋆	8⋆	7⋆	11⋆	5⋆	10⋆	10⋆	19⋆
5	4	6	8	8	11	7	7	7	15
4⋆	9⋆	5	6⋆	10⋆	10⋆	12⋆	4	4⋆	21⋆
3	10	2	6	10	4	15	1	3	20
6⋆	2⋆	6⋆	6⋆	8⋆	13⋆	2⋆	6⋆	5⋆	20⋆
7	1	8	8	8	14	1	3	7	22
1⋆	2	8⋆	3	7⋆	4⋆	2	4⋆	5	12⋆
3	2	7	0	7	6	3	7	0	13
2	5	1	5⋆	9⋆	4	6	2	2⋆	4⋆
3	4	0	8	9	6	4	0	6	3

TAB. A.43 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes de précédence est ajoutée.*

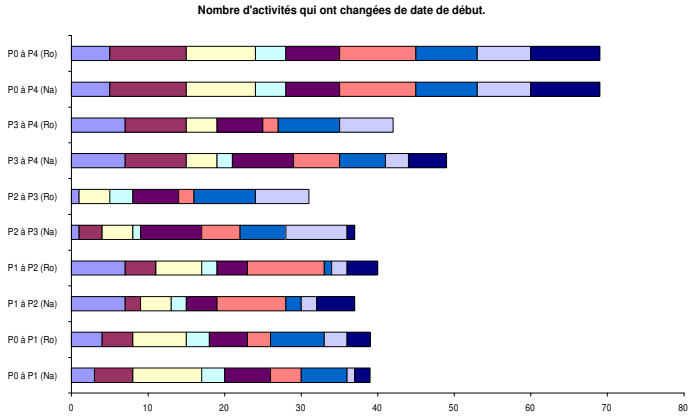


FIG. A.67 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

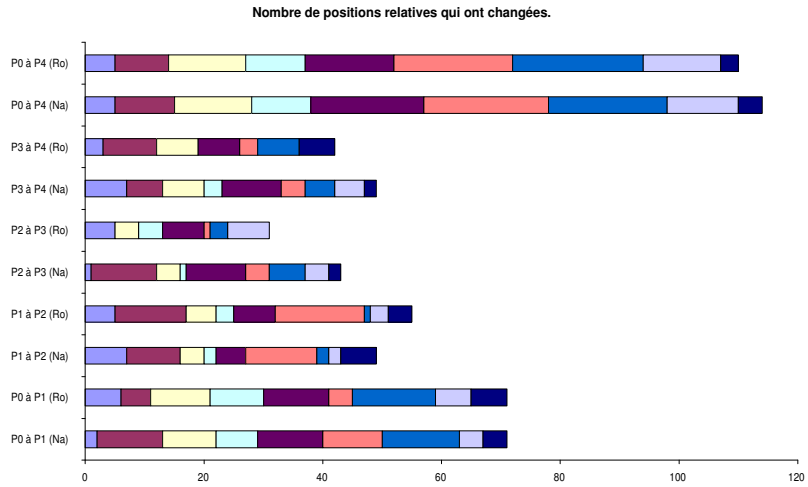


FIG. A.68 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

SDif.					**MDif.**				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
16	34⋆	3	41⋆	40⋆	10	16⋆	3	13⋆	15⋆
27	27	18	30	40	9	18	10	15	11
26	20	22	33⋆	39⋆	13	13	13	9⋆	10⋆
7	28	0	34	39	3	13	0	10	10
24⋆	13	14⋆	16	37⋆	5⋆	4	7⋆	6	9⋆
18	17	14	16	37	4	5	7	6	9
19⋆	3	2⋆	10	28⋆	16⋆	2	2⋆	9	16⋆
26	10	12	0	28	16	9	9	0	16
32⋆	23⋆	55⋆	55⋆	93⋆	10⋆	14⋆	11⋆	20⋆	26⋆
36	33	41	37	71	10	14	11	11	23
40⋆	115⋆	27	29⋆	189⋆	16⋆	20⋆	10	5⋆	31⋆
16	150	8	30	190	10	26	4	5	31
67⋆	11⋆	21⋆	24⋆	109 ⋆	18⋆	8⋆	9⋆	6⋆	30⋆
70	8	24	39	115	18	8	9	9	30
15⋆	10	34⋆	17	60⋆	15⋆	7	10⋆	9	18⋆
25	12	32	0	63	15	9	10	0	18
11	17	3	5⋆	14⋆	10	8	3	1⋆	3⋆
17	10	0	19	10	15	3	0	10	2

TAB. A.44 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes de précédence est ajoutée.*

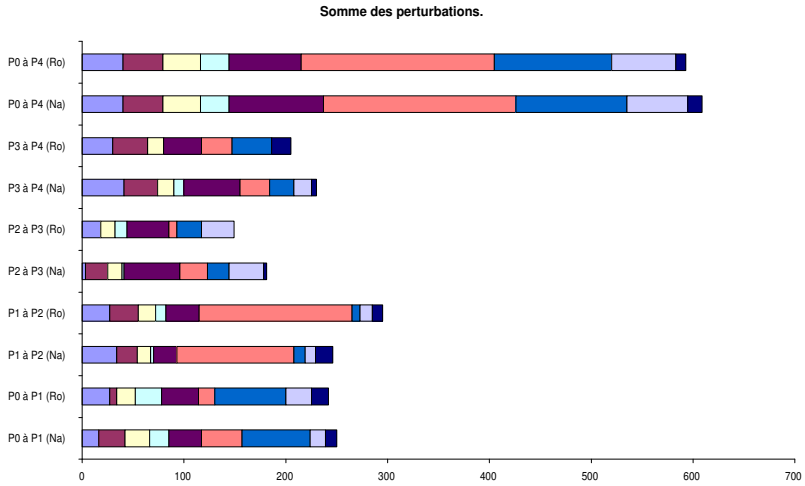


FIG. A.69 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

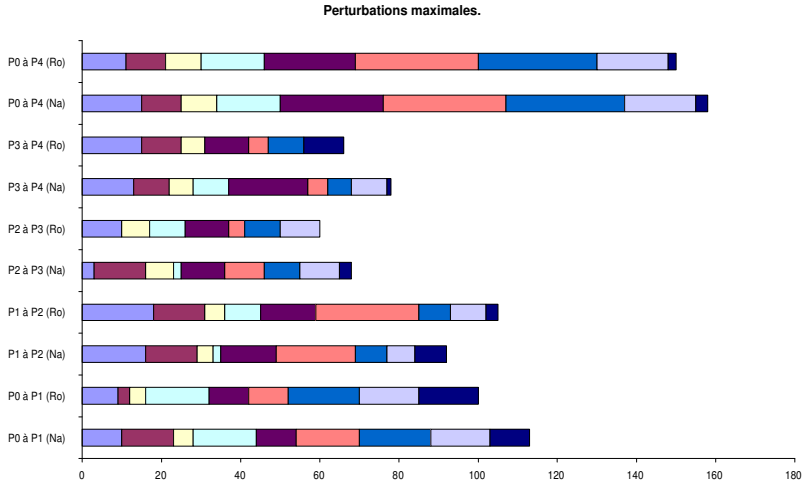


FIG. A.70 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série de contraintes de précédence
Problèmes comportant 22 activités et 4 ressources

Diff.			Posi.		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
2	15	16	3	52	52
14	8	15	33	20	22
17	5	14	32	13	31
14	5	16	33	7	37
4	10	12	7	20	24
1	15	15	4	28	32
13	11	9	21	20	30
11	12	9	28	9	30
15	19	16	19	31	22
16	18	17	27	31	39
10	3	10	33	11	38
10	4	9	37	8	39
1	4	4	6	15	20
5	5	7	8	15	20
13	5	10	19	9	16
16	11	9	36	41	15
12	6	13	18	9	15
14	5	13	13	8	12
7	0	7	20	0	20
13	9	14	39	12	46

TAB. A.45 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes de précédence est ajoutée.*

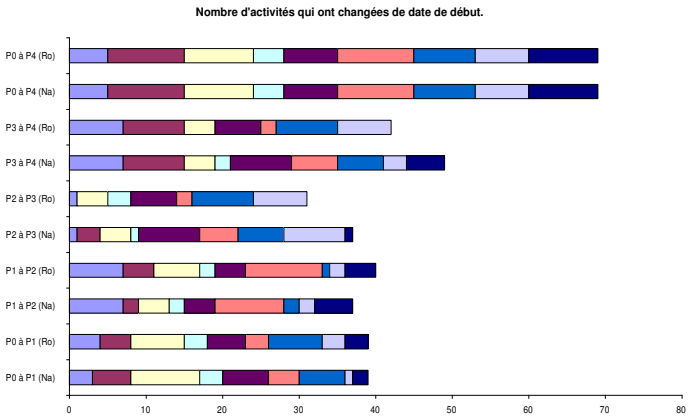


FIG. A.71 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

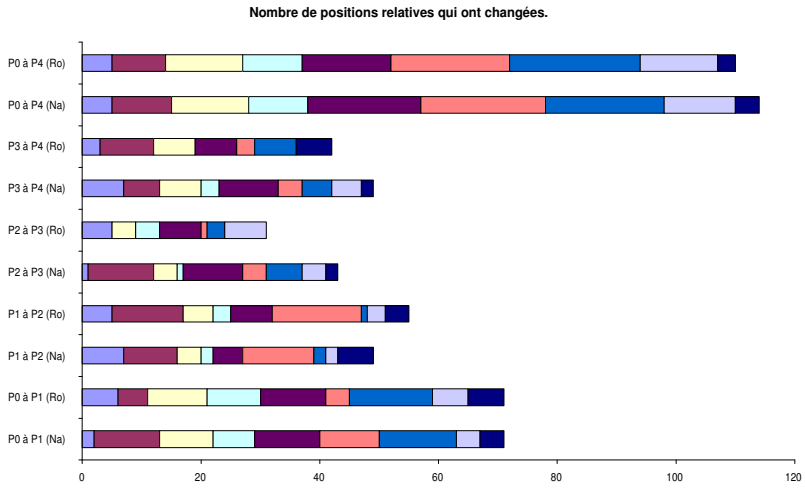


FIG. A.72 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

SDif.			MDif.		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
6	86	88	4	25	25
48	27	33	11	13	9
71	16	67	14	7	14
63	9	68	14	5	14
7	36	41	4	14	14
4	54	58	4	14	14
33	28	47	8	10	10
37	20	47	9	6	10
36	77	59	8	12	9
53	85	92	8	21	14
65	21	68	21	9	21
74	10	70	20	7	21
14	18	32	14	8	18
19	18	33	14	8	18
38	12	30	9	3	11
73	66	27	26	26	9
27	10	27	6	4	6
29	14	27	6	6	6
24	0	24	8	0	8
72	12	76	13	3	14

TAB. A.46 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes de précédence est ajoutée.*

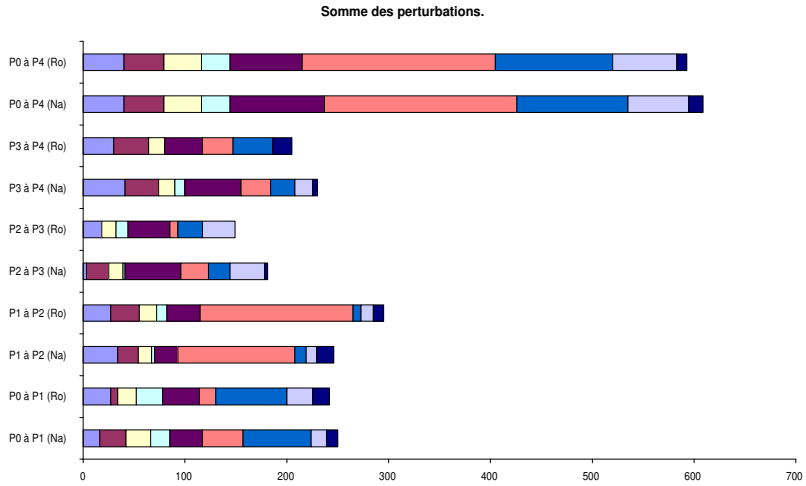


FIG. A.73 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

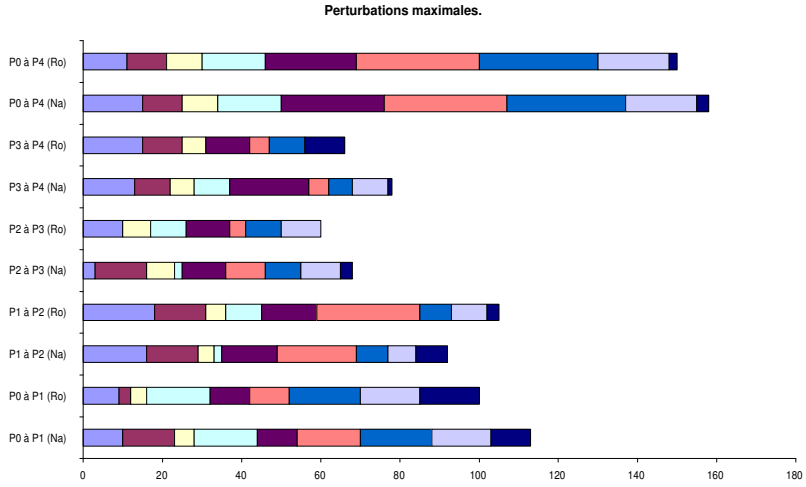


FIG. A.74 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est ajoutée.*

Résultats obtenus pour la stabilté
Ajout d'une série de contraintes de précédence
Problèmes comportant 32 activités

Diff.				**Posi.**			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
7	4	9	10	38	6	14	40
9	10	5	12	41	28	23	45
25⋆	13	10	27⋆	56⋆	43	23	90⋆
23	7	2	28	70	55	11	89
3	1	4	6	26	3	18	44
3	1	4	6	26	3	18	44
2	2	2	4	20	13	8	39
1	1	2	4	19	12	8	39
3	16⋆	3	18⋆	8	20⋆	14	36⋆
1	19	9	22	5	28	18	45
5	11	6	15	6	34	32	48
5	6	10	12	6	32	44	63
3	4	5	8	32	14	32	66
3	5	4	5	31	29	29	61
5	5	2	12	16	13	15	44
10	8	3	10	46	68	18	63
4	13	1	15	25	19	9	49
13	17	1	14	43	25	19	73
3	4	4	7	10	3	19	28
4	4	5	5	14	5	13	22
24	6	6	25	90	23	44	120
22	4	7	24	101	38	43	142
2	6	2	7	6	16	7	23
11	11	5	7	23	20	10	27
11	10	8	19	43	41	36	73
14	6	2	20	70	12	12	82
2	4	10	14	5	14	56	71
12	13	9	14	25	39	65	70
14	17	2	14	54	19	3	66
18	14	0	15	81	55	0	87
8	10	5	16	26	25	6	43
18	10	0	19	92	35	0	82
5	6	7	11	8	32	35	60
4	5	5	11	7	30	32	58
1	14	20	20	7	22	127	128
1	12	20	20	7	23	125	131
11	8	20⋆	19⋆	17	4	28⋆	36⋆
5	0	21	21	11	0	63	71
25⋆	12	10	23⋆	45⋆	29	26	48⋆
22	10	13	23	84	14	69	82

TAB. A.47 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est ajoutée.*

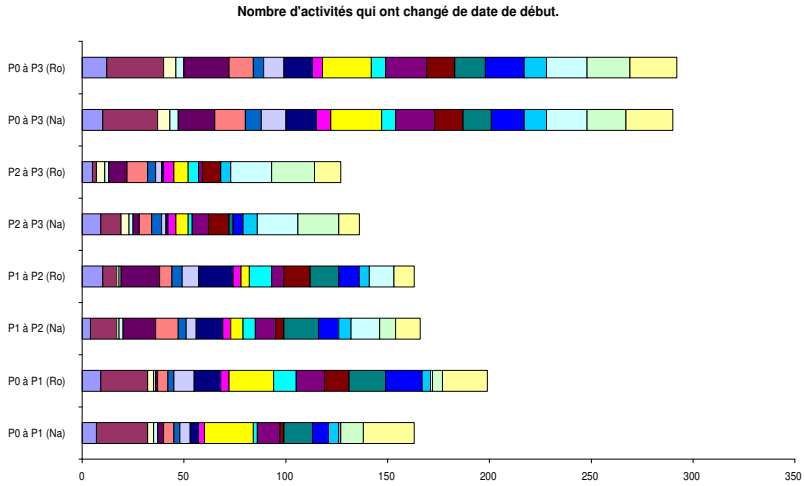


FIG. A.75 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est ajoutée.*

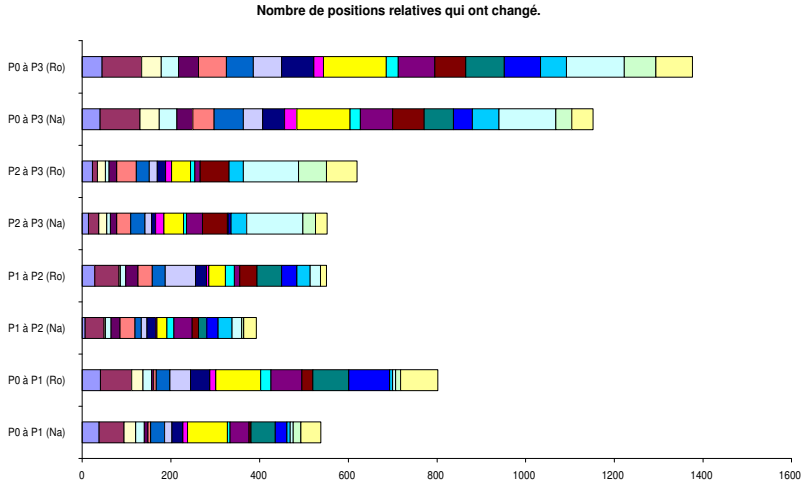


FIG. A.76 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est ajoutée.*

SDif.				MDif.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
55	5	32	70	34	2	9	30
65	36	24	67	27	10	10	27
104⋆	42	28	144⋆	11⋆	11	8	15⋆
116	49	8	155	14	11	5	15
33	2	16	49	22	2	10	21
33	2	16	49	22	2	10	21
31	16	8	51	29	14	6	29
29	14	8	51	29	14	6	29
5	39⋆	18	56⋆	3	7⋆	6	7⋆
3	49	24	74	3	16	9	16
7	44	36	75	3	13	10	23
8	46	58	90	3	12	11	23
31	9	28	54	20	5	13	16
27	31	18	52	18	13	14	15
17	14	11	42	11	5	10	11
48	81	12	73	11	23	10	23
30	23	9	54	23	7	9	22
67	33	17	77	27	3	17	25
16	5	25	42	9	2	9	10
26	6	13	33	10	2	7	10
169	44	78	225	24	36	41	39
201	60	72	289	33	44	49	47
11	28	9	32	10	7	7	7
42	41	11	36	12	12	7	8
36	42	24	72	9	10	6	8
58	22	7	87	10	6	6	10
4	13	54	71	2	10	19	19
22	46	68	66	4	12	27	19
100	35	11	114	17	9	6	18
139	73	0	136	17	18	0	18
28	32	8	58	7	5	2	7
125	34	0	113	23	10	0	23
7	38	41	66	2	21	18	21
5	36	36	65	2	21	18	21
7	21	271	281	7	4	25	24
7	19	263	275	7	4	22	24
24	12	55⋆	61⋆	5	5	19⋆	19⋆
14	0	122	132	5	0	19	19
94⋆	48	45	93⋆	27⋆	15	15	27⋆
142	26	103	165	27	4	26	27

TAB. A.48 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est ajoutée.*

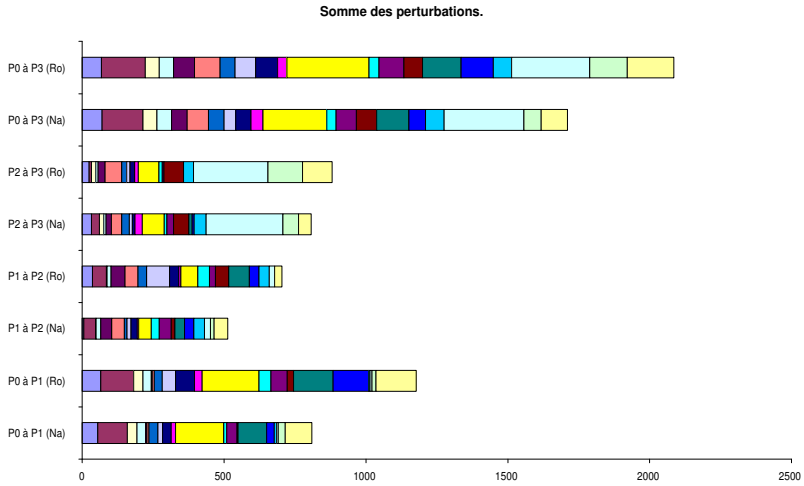


FIG. A.77 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est ajoutée.*

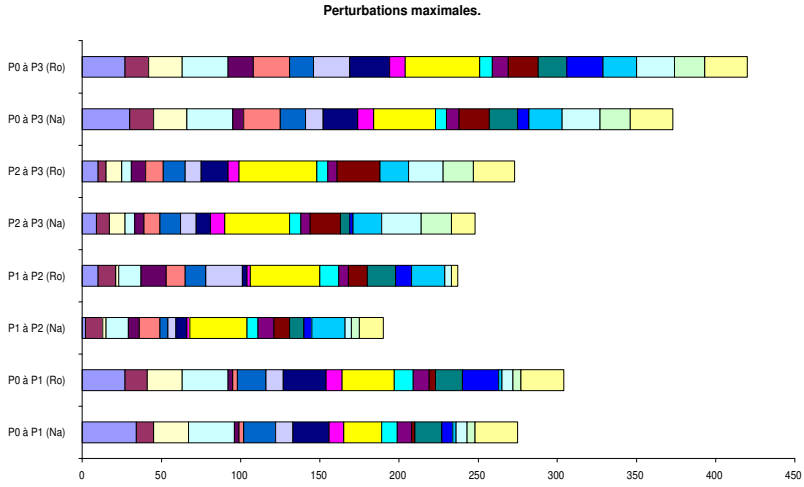


FIG. A.78 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est ajoutée.*

Résultats obtenus pour la stabilté
Retrait d'une série de contraintes de précédence
Problèmes comportant 12 activités et 4 ressources

Diff.			Posi.		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
5	9⋆	8⋆	9	10⋆	11⋆
6	7	8	11	13	12
3	2	3	6	4	7
4	5	3	7	5	10
4	3	4	16	11	10
2	2	3	8	2	9
7⋆	6⋆	5⋆	5⋆	9⋆	9⋆
7	9	7	5	10	10
7⋆	3	6⋆	15⋆	6	15⋆
7	2	6	14	5	15
5⋆	5	5⋆	7⋆	9	14⋆
4	2	4	6	7	11
5	7	7	10	7	12
3	6	7	9	6	12
1	2⋆	2⋆	4	6⋆	8⋆
1	2	2	7	6	6
9⋆	4⋆	8⋆	14⋆	6⋆	14⋆
8	2	8	13	1	14
4	4⋆	6⋆	8	7⋆	13⋆
5	5	6	5	9	14

TAB. A.49 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est retirée.*

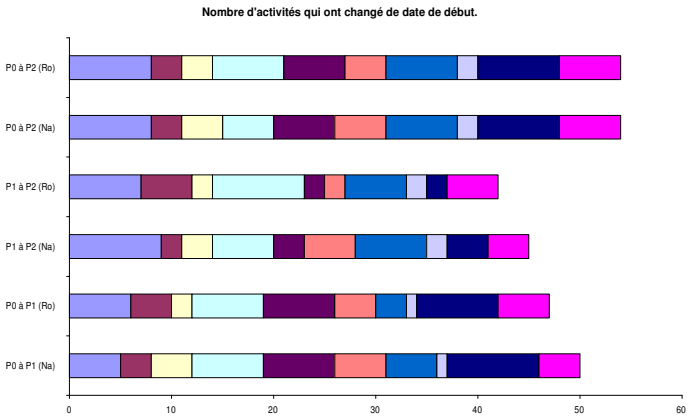


FIG. A.79 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

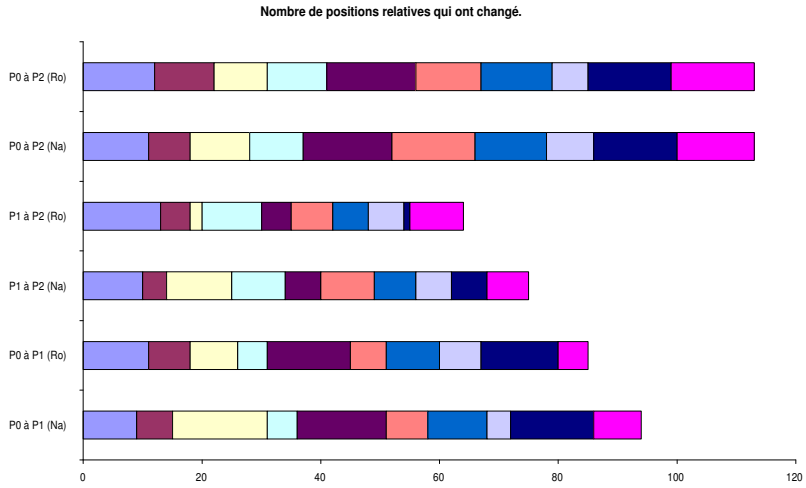


FIG. A.80 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

SDif.			**MDif.**		
$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$	$P_0 \to P_1$	$P_1 \to P_2$	$P_0 \to P_2$
32	37⋆	35⋆	9	7⋆	10⋆
47	43	44	11	14	12
20	11	19	9	6	9
31	27	34	11	9	16
34	21	19	17	13	12
19	2	21	17	1	18
13⋆	19⋆	16⋆	4⋆	9⋆	9⋆
13	29	20	4	9	9
66⋆	17	51⋆	25⋆	10	25⋆
64	15	51	25	10	25
27⋆	34	53⋆	14⋆	20	20⋆
25	21	42	14	19	19
60	84⋆	126⋆	26	26⋆	29⋆
51	75	126	26	26	29
7⋆	21⋆	28⋆	7⋆	17⋆	24⋆
24	22	18	24	14	10
55	19	64	18	10	18
61	3	64	18	2	18
16⋆	16	30⋆	9	6⋆	15⋆
16	17	29	8	7	15

TAB. A.50 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes de précédence est retirée.*

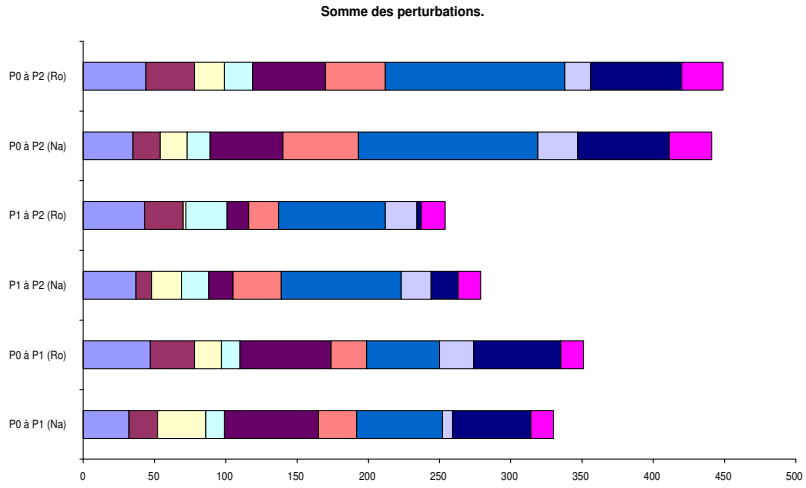


FIG. A.81 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

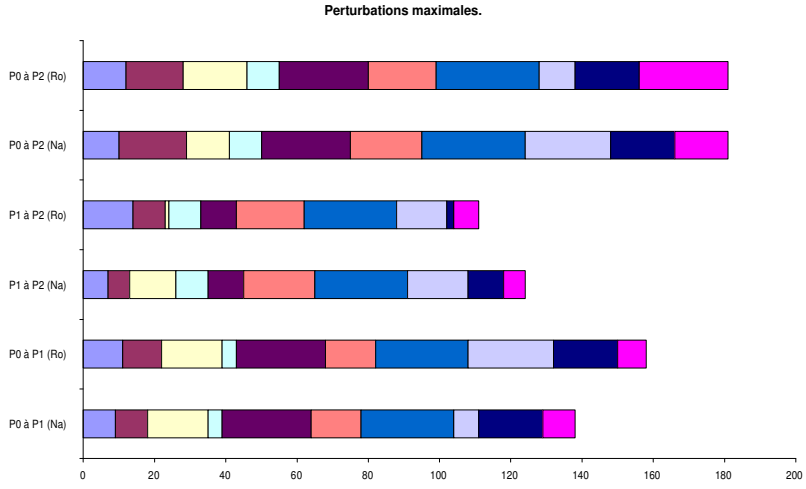


FIG. A.82 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

Résultats obtenus pour la stabilté

Retrait d'une série de contraintes de précédence

Problèmes comportant 22 activités et 4 ressources

Diff.				Posi.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
7	13⋆	1	11⋆	7	9⋆	1	9⋆
6	12	5	12	10	13	8	17
12⋆	15⋆	2	14⋆	26⋆	20⋆	12	42⋆
13	13	2	14	31	7	11	42
4	9	1	10	17	34	6	39
10	2	2	9	38	10	8	37
7	9⋆	8	144⋆	5	27⋆	24	31⋆
8	12	1	12	6	29	9	29
7	11	12	11	28	31	29	28
1	6	8	11	9	9	22	31
3	2	2	7	18	5	3	26
6	5	4	8	22	8	5	28
9	5	5	8	14	12	9	24
9	3	4	7	15	12	13	28
2	8	4	10	3	25	6	29
3	2	2	5	5	12	4	21
9	12	15⋆	14⋆	13	21	40⋆	41⋆
7	9	15	15	16	19	19	28
16	10⋆	10⋆	16⋆	46	17⋆	19⋆	44⋆
5	13	9	18	18	40	17	51

TAB. A.51 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes de précédence est retirée.*

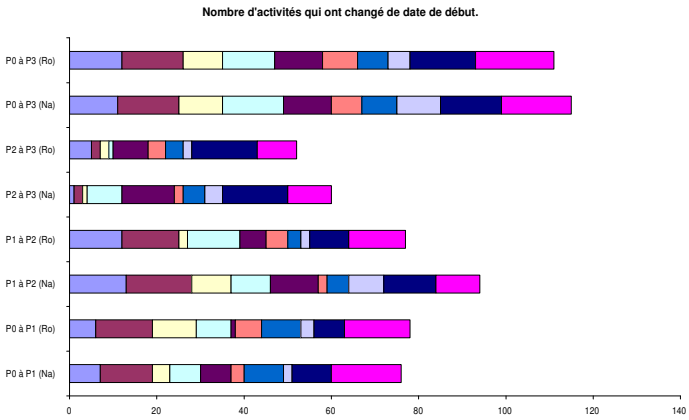


FIG. A.83 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

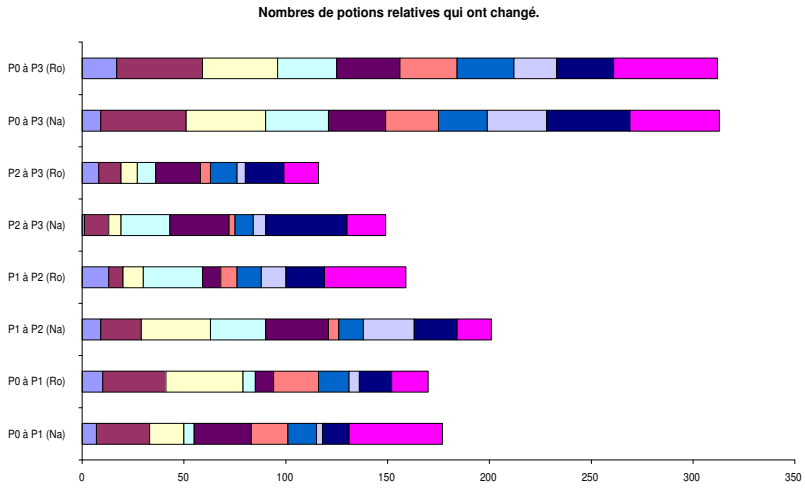


FIG. A.84 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

SDif.				MDif.			
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_0 \to P_3$
11	20⋆	2	17⋆	3	3⋆	2	3⋆
13	20	11	24	4	5	5	4
36⋆	38⋆	13	65⋆	9⋆	12⋆	12	12⋆
40	24	10	62	8	12	9	12
31	70	6	57	16	14	6	13
61	14	8	67	13	12	7	13
10	47	42	51	4	13	20	13
11	44	10	55	4	13	10	13
43	62	62	47	15	13	13	14
16	14	37	53	16	5	8	16
29	6	6	41	26	5	3	26
35	11	8	44	26	7	3	26
21	16	17	34	8	9	9	13
23	16	20	37	8	10	10	14
2	48	12	48	1	24	4	24
4	26	6	32	2	24	3	24
25	35	62⋆	70⋆	7	6	14⋆	13⋆
28	33	44	55	13	14	14	13
73	23⋆	27⋆	71⋆	14	9⋆	9⋆	13⋆
22	66	25	85	14	13	6	13

TAB. A.52 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes de précédence est retirée.*

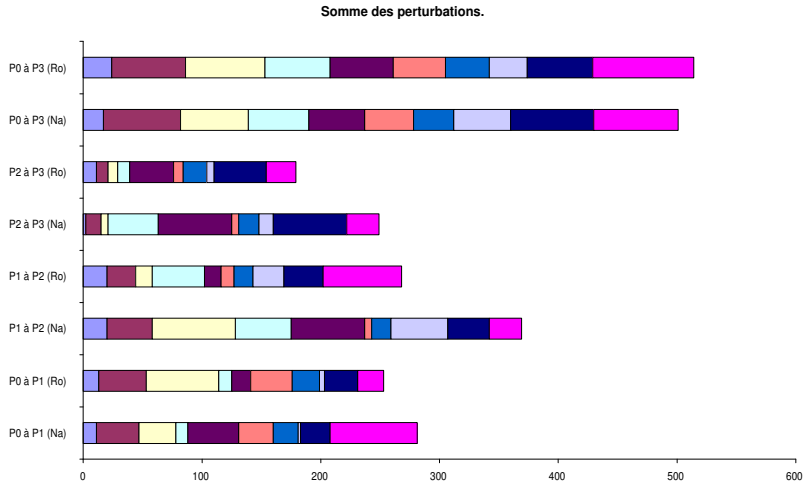


FIG. A.85 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

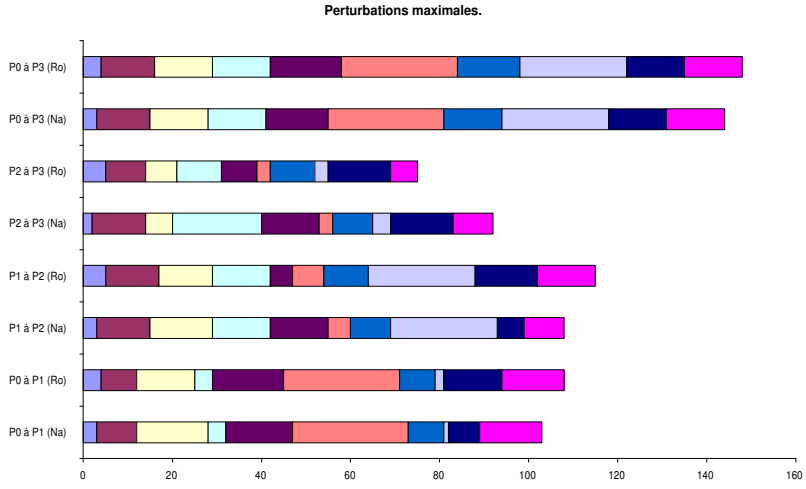


FIG. A.86 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes de précédence est retirée.*

Résultats obtenus pour la stabilté

Retrait d'une série de contraintes de précédence

Problèmes comportant 32 activités

Diff.					**Posi.**				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
2	8	16	6	17	20	46	85	19	78
4	13	1	1	14	7	65	10	3	83
11	16	13	15	15	36	32	43	38	62
9	13	14	13	19	20	22	27	35	61
15	1	14	15	17	52	7	30	44	80
9	9	5	7	13	25	22	12	25	53
2	5	1	9	12	6	27	22	36	90
4	12	2	15	16	15	40	19	77	113
2	7	1	13	18	2	42	10	59	51
2	2	3	19	20	13	28	17	72	99
4	5	2	1	9	7	10	1	2	18
5	1	4	17	16	58	5	8	89	81
1	3	1	5	10	4	28	1	6	38
5	15	2	4	13	8	57	9	24	72
2	22⋆	14	15	25⋆	2	49⋆	51	38	85⋆
4	25	12	9	25	14	62	56	28	84
13	14⋆	15	11	20⋆	71	23⋆	78	48	131⋆
1	16	13	11	20	21	42	81	63	132
7	1	1	8	16	16	2	13	11	38
11	10	5	6	18	22	32	30	25	81
6	14	12	3	15	12	42	46	12	81
2	5	10	5	12	14	21	37	26	74
3	5	3	14	16	6	9	15	50	65
17	10	7	1	20	73	70	45	8	114
10	9	1	4	13	26	27	8	13	45
11	11	16	1	17	25	46	54	8	89
1	4	7	13	15	1	20	22	39	54
1	4	5	10	14	1	19	19	32	50
12	8	3	16	17	52	45	7	42	85
3	15	11	2	16	29	64	63	13	106
2	7	5	1	9	5	29	23	6	34
8	12	12	3	15	18	25	27	17	36
5	3	3	1	7	23	3	12	4	31
1	4	2	6	8	12	9	10	21	37
6	13	2	6	10	25	35	8	17	62
3	9	8	3	15	23	47	50	16	109
10	6	1	7	15	35	53	12	74	83
18	13	12	14	17	69	43	36	108	108
5	7	8	3	8	14	25	16	3	40
1	4	6	2	10	11	23	20	3	42

TAB. A.53 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est retirée.*

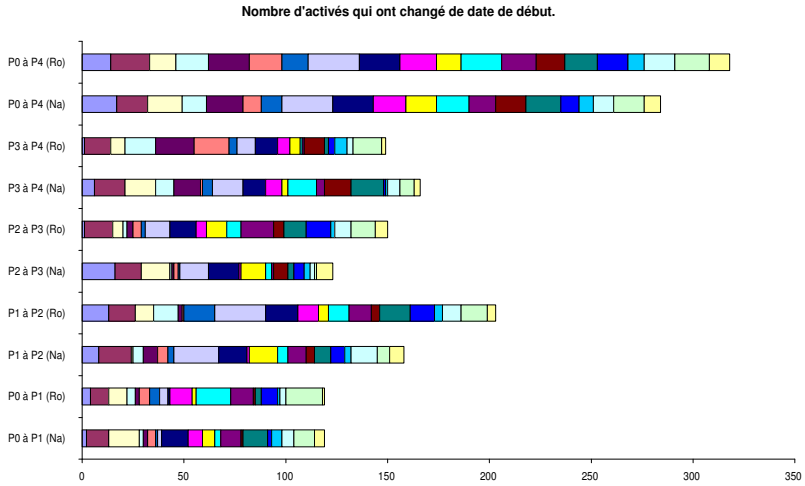


FIG. A.87 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est retirée.*

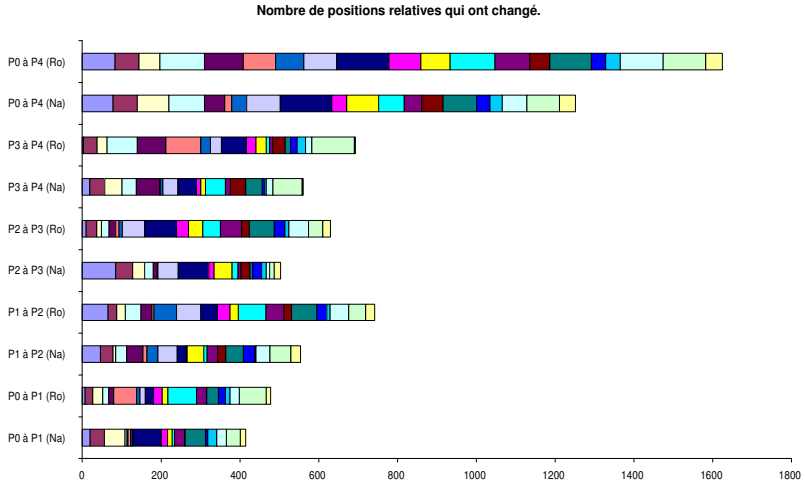


FIG. A.88 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est retirée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
20	58	116	20	100	10	13	29	6	27
5	86	11	1	101	2	14	11	1	15
44	59	62	63	82	12	9	13	15	19
31	34	38	54	83	15	10	13	11	15
61	7	52	57	93	13	7	8	10	12
30	23	11	39	73	7	9	7	10	16
5	38	26	43	100	4	28	26	16	28
25	40	19	93	141	10	10	18	25	28
2	47	8	80	75	1	34	8	24	13
10	35	13	105	141	9	34	11	15	33
7	16	4	3	26	2	6	2	3	6
65	3	10	99	93	22	3	5	16	18
3	33	1	5	42	3	28	1	1	28
7	70	9	16	74	3	28	5	8	28
3	93⋆	52	44	134⋆	2	16⋆	13	9	16⋆
18	98	57	34	131	13	13	15	6	14
116	57⋆	150	84	285⋆	31	13⋆	20	27	32⋆
51	84	133	106	294	51	10	20	31	51
13	1	12	9	35	4	1	12	2	12
18	35	30	33	94	3	21	12	21	22
16	57	74	15	100	11	17	36	11	35
18	23	62	23	112	16	16	47	8	47
7	11	11	57	72	3	3	8	17	18
72	46	33	12	121	13	12	11	12	15
32	33	11	13	51	14	10	11	8	14
42	47	76	5	124	14	16	11	5	20
2	13	20	32	47	2	6	11	6	7
2	14	17	33	50	2	6	11	6	8
41	32	4	31	72	10	13	2	4	12
21	58	56	5	104	16	13	10	3	16
6	48	30	7	51	5	21	7	7	21
17	44	29	20	60	6	21	8	18	16
19	3	9	2	25	10	1	4	2	10
10	6	6	23	35	10	3	5	18	15
21	30	8	15	56	8	11	7	10	11
24	50	63	15	130	18	22	20	13	28
34	74	14	119	117	10	17	14	31	31
107	82	55	160	154	14	14	12	37	31
15	27	31	6	41	9	12	7	4	12
9	21	26	5	53	9	12	7	4	13

TAB. A.54 – *RRésultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes de précédence est retirée.*

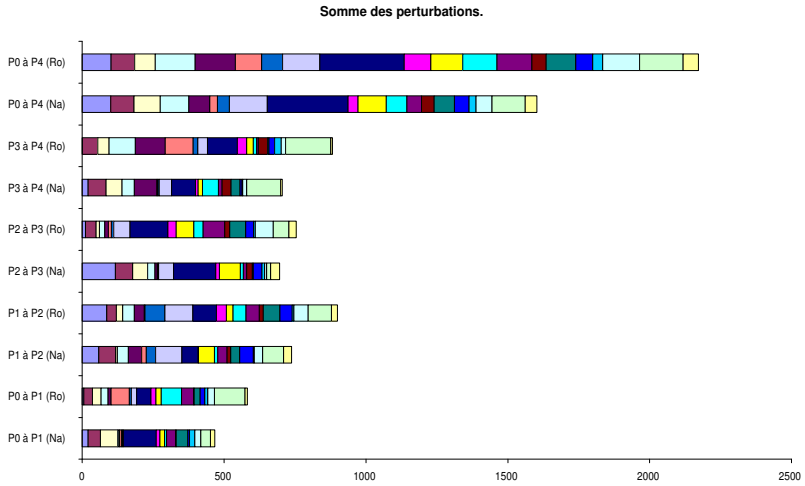


FIG. A.89 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est retirée.*

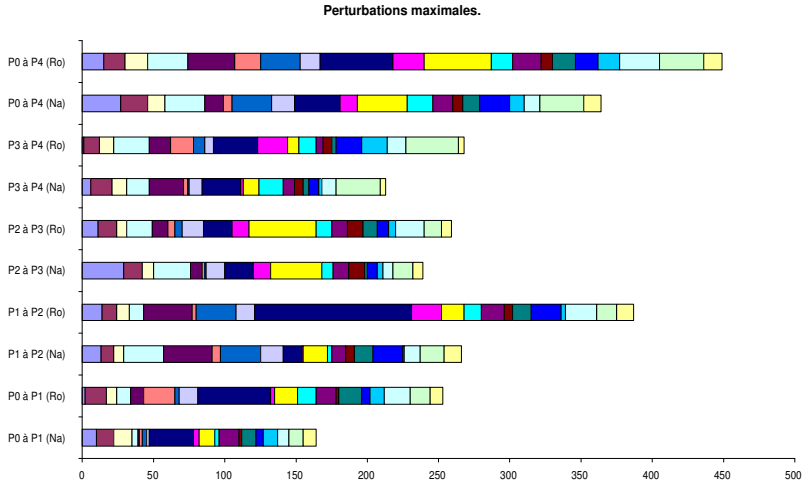


FIG. A.90 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes de précédence est retirée.*

Annexe B

Résultats des tests : contraintes temporelles généralisées

Dans cette annexe, nous reportons les tableaux des résultats obtenus lors de nos expérimentations pour la résolution du RCPSP dynamique lorsque nous ajoutons une série de contraintes temporelles généralisées.

Les notations que nous utilisons sont les mêmes que ceux présentées dans l'annexe A (page 194).

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série de contraintes temporelles généralisées
Problèmes comportant 12 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		**moy.**	**min.**	**max.**
25.60	40.5%	-4.91	26.8%	-5.13	19.8%	-12.85	3.4%	15.37	6.33	30.39
								15.91	1.20	32.77
17.93	32.4%	-0.26	28%	-0.67	23.9%	-21.18	-5.3%	16.32	7.97	29.40
								15.49	6.46	29.40
-1.09	-3.8%	29.61	43.1%	4.05	31.2%	26.24	42%	16.23	7.97	34.20
								27.99	13.72	38.25
1.63	85%	134.05	76.6%	- 4.40	69%	-8.34	62.7%	14.58	13.11	17.28
								39.16	5.86	147.16
0.66	5.5%	3.17	21.9%	1.04	24.5%	4.36	35.2%	3.38	1.33	5.95
								5.23	2.37	6.61
4.02	34.5%	-2.55	7.5%	-1.91	-1.6%	-1.43	-5.5%	7.06	2.23	10.38
								6.69	5.37	7.83
1.52	8.8%	4.51	21.8%	2.95	21.8%	2.89	22.7%	8.07	6.05	10.48
								10.45	7.66	13.43
4.88	26.7%	10.37	48.5%	2.93	40.6%	3.19	37.9%	6.98	2.77	10.46
								11.25	7.69	13.39
7.27	31.6%	6.31	43.5%	3.26	45.1%	3.85	47.2%	4.61	1.94	11.43
								8.75	6.06	11.52
2.61	17.8%	7.05	34.2%	-5.46	11.9%	1.39	13.5%	7.15	4.28	12.46
								8.27	6.18	13.60
7.47	32.5%	-0.42	22.9%	3.74	28.5%	2.07	28.2%	6.52	3.38	11.71
								9.09	7.12	11.71
-0.38	-12.1%	0.23	-3.4%	0.80	11.9%	0.49	17.1%	1.09	0.34	2.04
								1.32	1.08	2.04
0.72	15.5%	-2.28	-22.3%	-0.77	-29.2%	0.39	-21.7%	2.17	0.58	4.64
								1.78	0.97	2.50
2.77	15.4%	0.97	12.9%	6.08	24.8%	0.48	23.5%	6.69	3.82	10.00
								8.75	4.30	10.97
9.48	24.4%	6.53	26.9%	1.12	23.5%	-0.81	19.9%	13.06	9.60	14.76
								16.33	8.79	24.03
moy.	23.6%		25.8%		23%		21.3%	30.6%		

TAB. B.1 – *Résultats pour le gain en temps pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes temporelles généralisées est ajoutée.*

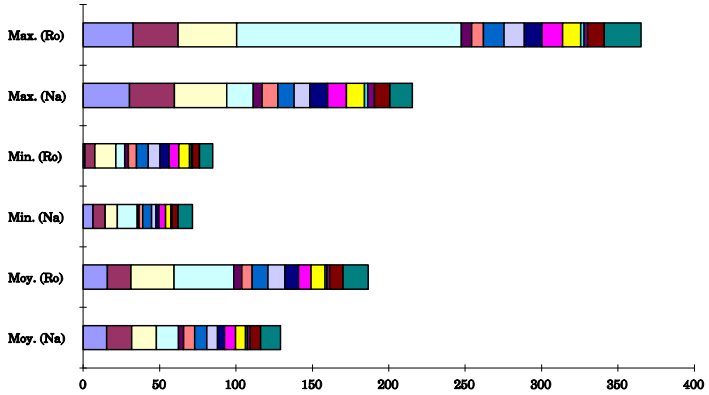


FIG. B.1 – *Représentation de quelques résultats du tableau B.1.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série de contraintes temporelles généralisées
Problèmes comportant 22 activités et 4 ressources

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		$P_4 \rightarrow P_5$		**moy.**	**min.**	**max.**
348.88	51.1%	-130.44	27.5%	-14.94	24.8%	0.15	24.3%	0.51	24.1%	128.08	8.38	240.64
										168.81	8.89	469.05
4.30	1.5%	11.96	4.1%	9.45	5.4%	-2.48	4.1%	14.84	5.6%	126.84	66.56	139.50
										131.49	76.01	139.50
66.82	14%	253.96	26.9%	327.55	40.3%	109.76	39.4%	464.11	50.1%	243.23	53.96	460.82
										394.85	225.13	714.78
1.29	0.5%	0.26	0.6%	4.80	2.5%	5.87	4.6%	8.09	7.3%	51.24	4.97	214.35
										53.68	6.26	214.35
385.07	61.7%	-8.55	54%	1.09	32.9%	169.04	37%	233.94	43.3%	204.36	80.60	449.44
										313.69	72.05	484.71
3.31	5.1%	8.11	10%	-1.42	7.5%	10.05	13.5%	4.49	15%	27.71	6.19	41.69
										31.72	15.07	49.80
5.44	8.3%	13.13	18.7%	78.51	51.1%	224.24	50.5%	113.96	47.2%	97.38	11.99	221.22
										161.64	32.52	445.46
0.31	1.1%	4.28	14.3%	1.45	15.2%	-0.59	11.3%	-0.33	9.2%	10.03	1.14	21.76
										11.12	4.90	21.76
-0.61	-2.3%	1.04	1.3%	0.05	1.2%	5.52	10.7%	4.33	12.6%	14.23	3.61	21.76
										15.43	4.50	25.87
0.99	3.7%	5.16	13%	144.42	58.4%	40.88	58.9%	178.83	63.4%	42.61	3.38	79.97
										80.90	4.37	258.80
3.51	35.4%	1.34	31.6%	22.72	14%	-0.49	9.5%	10.28	9.7%	69.05	1.47	158.72
										74.47	4.91	181.44
3.94	40.6%	4.23	56.3%	3.99	62.6%	34.06	9.2%	73.80	16.6%	119.93	0.58	446.19
										129.18	4.77	480.25
moy.	18.3%		21.5%		26.3%		22.7%		25.3%	27.5%		

TAB. B.2 – *Résultats pour le gain en temps pour des séries de problèmes comportant 22 activités et 4 ressources. Le cas où une série de contraintes temporelles généralisées est ajoutée.*

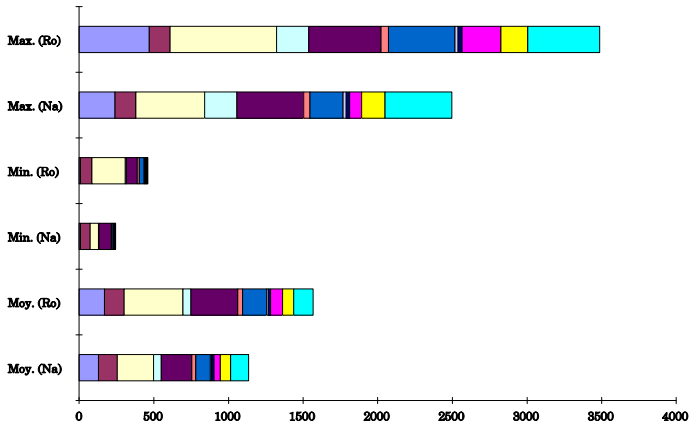


FIG. B.2 – *Représentation de quelques résultats du tableau B.2.*

Résultats obtenus pour le temps de calcul CPU

Ajout d'une série de contraintes temporelles généralisées

Problèmes comportant 32 activités

$P_0 \rightarrow P_1$		$P_1 \rightarrow P_2$		$P_2 \rightarrow P_3$		$P_3 \rightarrow P_4$		$P_4 \rightarrow P_5$		moy.	min.	max.
9.86	42.8%	-0.78	12.6%	10.05	16.6%	5.67	16.6%	-1.59	13.6%	29.45	1.31	49.65
										34.41	11.17	48.87
4.80	17.1%	79.29	51%	31.80	47.5%	46.92	48.6%	114.79	56.4%	42.83	11.47	57.49
										75.40	11.72	156.99
26.59	18.6%	-78.24	-23.2%	80.18	9.3%	54.25	21.9%	41.20	28.1%	63.33	1.56	157.60
										79.89	63.54	81.74
14.20	37.9%	27.48	56.8%	9.22	47.1%	28.27	56.8%	19.35	57.7%	14.43	2.99	25.36
										30.27	15.85	35.84
11.17	16.9%	-25.56	-14.4%	2.56	-9.3%	-35.76	-35.1%	20.82	-14.5%	42.17	17.77	59.44
										32.65	8.86	48.88
-4.37	-9.3%	-0.50	-6.9%	20.45	16.8%	24.53	33.8%	1.67	22.6%	28.57	1.79	64.33
										36.59	20.59	66.00
13.27	24.6%	-51.98	-38.4%	43.79	3.4%	-35.37	-15.8%	822.82	38.5%	253.19	2.48	1044.31
										247.13	25.71	1867.13
-65.35	-7.6%	77.77	0.8%	16.52	1.5%	47.64	4%	35.44	5.7%	367.16	4.77	590.05
										382.48	52.41	554.63
84.05	22.2%	62.91	26.6%	78.69	34.3%	-4.13	29%	-17.49	23.3%	133.65	26.58	212.02
										177.95	105.27	212.02
-202.20	-51.7%	-52.84	-41.2%	37.63	-26.3%	302.71	6.3%	190.51	16.5%	278.46	132.12	345.51
										295.52	143.31	519.98
5.28	6.2%	5.03	8.6%	60.84	34.4%	19.64	24%	30.89	30.3%	44.00	25.37	42.92
										54.31	5.73	88.25
-21.51	-26.4%	0.41	-19.1%	3.78	-12.3%	10.04	-4.7%	-5.90	-7.8%	36.41	2.10	94.65
										34.95	8.10	73.14
-32.54	-61.3%	6.67	-31.7%	8.03	-16.1%	-16.81	-25.5%	1.26	-21.1%	38.19	20.84	62.84
										31.26	22.10	30.30
-33.21	-10.8%	31.08	-0.5%	-66.64	-17.3%	30.10	-8.8%	-46.51	-18.3%	109.97	8.98	259.72
										102.24	29.68	259.72
-4.13	-13.1%	27.92	37.4%	9.81	38.1%	73.03	54.8%	62.23	57.4%	37.42	12.58	37.08
										58.75	9.78	106.50
moy.	0.4%		1.2%		11.1%		13.7%		19.2%	9.2%		

TAB. B.3 – *Résultats pour le gain en temps pour des séries de problèmes comportant 32 activités. Le cas où une série de contraintes temporelles généralisées est ajoutée.*

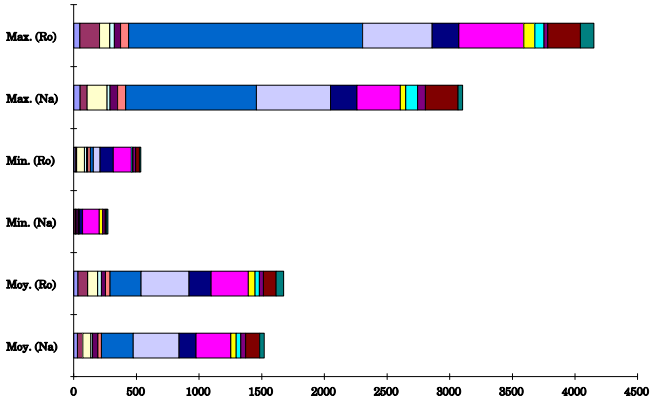


FIG. B.3 – *Représentation de quelques résultats du tableau B.3.*

Résultats obtenus pour la stabilté

Ajout d'une série de contraintes temporelles généralisées
Problèmes comportant 12 activités et 4 ressources

Diff.					Posi.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
2	9⋆	2	7⋆	9⋆	2	1⋆	1	2⋆	2⋆
2	8	3	7	9	3	6	5	0	4
1	10⋆	1	4⋆	9⋆	2	0⋆	1	0⋆	3⋆
1	10	1	3	9	2	0	1	1	4
6	4⋆	7	1	8⋆	8	6⋆	10	1	12⋆
6	7	7	0	8	13	10	10	0	12
1	7⋆	2	2	8⋆	3	10⋆	2	2	9⋆
1	5	6	4	8	3	6	9	5	9
7⋆	6⋆	4⋆	6⋆	7⋆	15⋆	14⋆	3⋆	1⋆	6⋆
7	5	3	6	7	9	8	2	4	8
1	5⋆	9	5	8⋆	1	1⋆	16	2	14⋆
1	4	9	2	9	1	0	17	3	15
3	1⋆	6⋆	7⋆	8⋆	6	1⋆	3⋆	0⋆	8⋆
3	3	6	7	8	6	2	1	0	8
3	3⋆	10⋆	5⋆	7⋆	6	2⋆	6⋆	1⋆	2⋆
3	1	10	6	7	6	1	6	1	3
3	4	9⋆	3	9⋆	4	1	0⋆	1	4⋆
2	4	9	3	9	3	2	0	1	4
7⋆	3⋆	10⋆	3	10⋆	3⋆	2⋆	4⋆	3	7⋆
7	3	10	6	10	3	4	5	6	8
3	6⋆	2	3	8⋆	4	3⋆	1	3	7⋆
2	8	4	0	8	3	6	4	0	7
6	3	3	3	7	7	5	1	7	9
5	1	1	1	7	5	2	1	0	8
5	4	2	1	6	6	4	2	2	10
2	1	6	3	6	1	2	5	4	10
4	8⋆	0	5⋆	8⋆	2	8⋆	0	5⋆	5⋆
4	8	3	6	8	2	8	5	6	5
3	1	4	3	8	3	3	3	1	7
7	4	4	4	8	6	6	4	3	8

TAB. B.4 – *Résultats pour les mesures* DIFF *et* POSI *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes temporelles généralisées est ajoutée.*

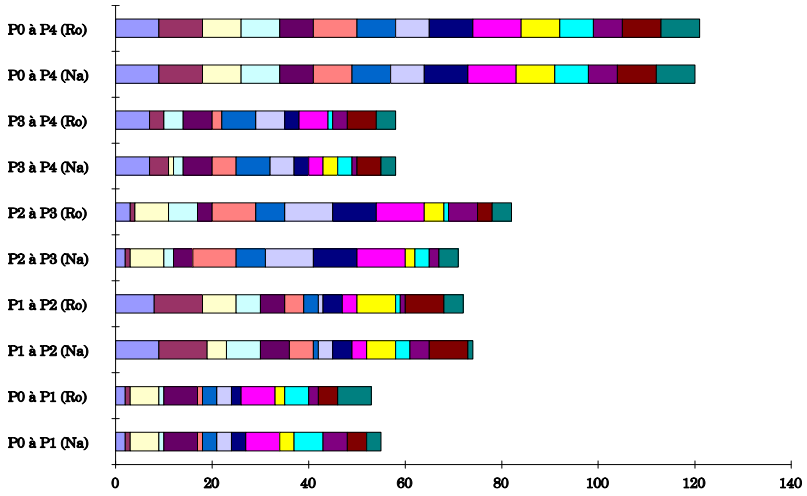


FIG. B.4 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

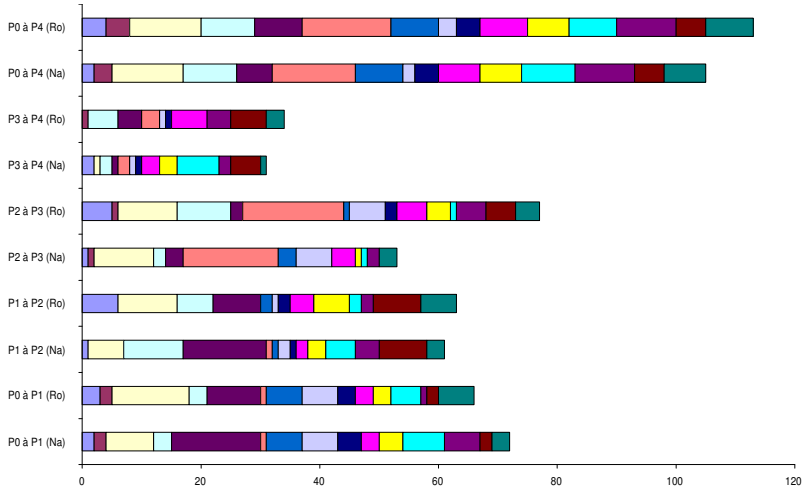


FIG. B.5 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

SDif.					MDif.				
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_0 \to P_4$
11	17⋆	3	19⋆	40⋆	7	2⋆	2	6⋆	9⋆
14	23	16	17	40	7	9	13	5	9
4	20⋆	2	12⋆	26⋆	4	2⋆	2	3⋆	5⋆
4	20	2	9	27	4	2	2	3	5
35	29⋆	45	1	54⋆	9	9⋆	12	1	13⋆
51	47	44	0	54	10	10	11	0	13
5	43⋆	10	10	48⋆	5	9⋆	9	9	9⋆
5	35	26	18	48	5	9	8	8	9
68⋆	64⋆	10⋆	7⋆	25⋆	19⋆	18⋆	5⋆	2⋆	7⋆
36	30	5	14	27	15	16	3	9	9
3	8⋆	103	17	85⋆	3	4⋆	16	10	15⋆
3	4	99	14	88	3	1	16	10	15
16	2⋆	16⋆	10⋆	36⋆	7	2⋆	6⋆	4⋆	11⋆
16	8	8	10	36	7	5	3	4	11
21	8⋆	35⋆	5⋆	25⋆	9	5⋆	9⋆	1⋆	8⋆
21	5	35	9	28	9	5	9	3	8
16	10	9⋆	6	27⋆	9	4	1⋆	4	8⋆
12	14	9	6	27	9	8	1	4	8
16⋆	8⋆	36⋆	10	64⋆	10⋆	5⋆	6⋆	5	14⋆
16	10	38	21	53	10	6	8	9	13
16	22⋆	5	12	41⋆	9	10⋆	4	10	10⋆
12	40	13	0	41	9	10	9	0	10
29	14	12	24	39	9	8	4	17	9
25	3	1	1	28	9	3	1	1	9
24	12	6	3	29	9	7	4	3	9
7	7	29	15	34	5	7	9	9	9
10	19⋆	0	11⋆	28⋆	5	5⋆	0	6⋆	6⋆
10	19	7	19	33	5	5	5	7	6
31	10	13	21	71	11	10	6	10	20
47	32	18	28	77	11	20	6	10	20

TAB. B.5 – *Résultats pour les mesures* SDIF *et* MDIF *pour des séries de problèmes comportant 12 activités et 4 ressources. Le cas où une série de contraintes temporelles généralisées est ajoutée.*

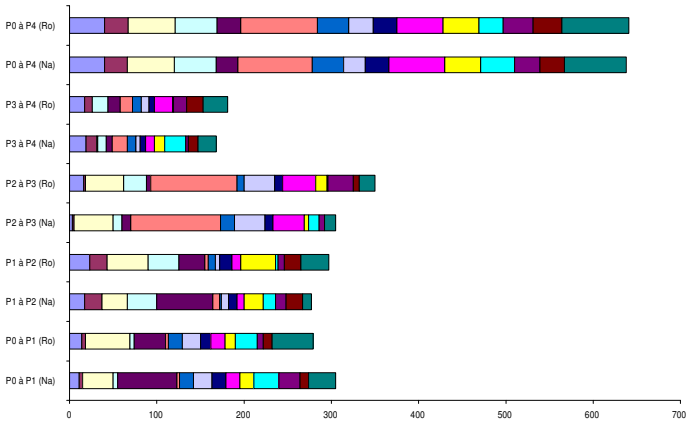


FIG. B.6 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

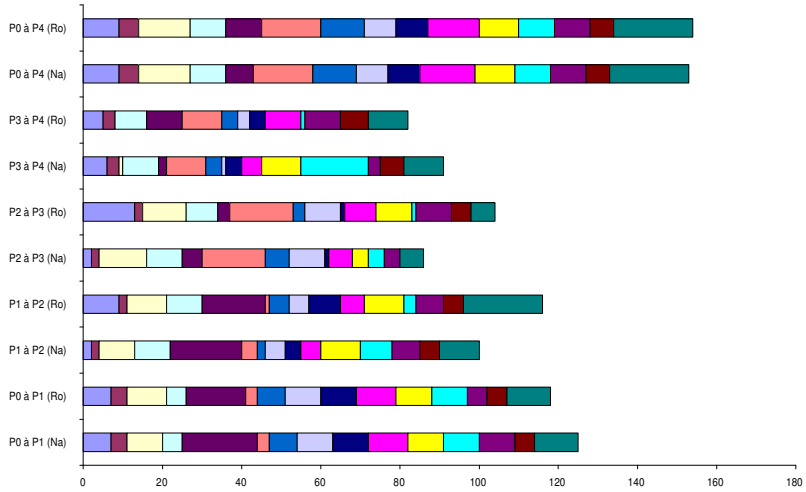


FIG. B.7 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 12 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

Résultats obtenus pour la stabilté

Ajout d'une série de contraintes temporelles généralisées
Problèmes comportant 22 activités et 4 ressources

Diff.						Posi.					
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_4 \to P_5$	$P_0 \to P_5$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_4 \to P_5$	$P_0 \to P_5$
7	3	4	1	1	7	20	1	2	2	1	20
5	12	12	12	9	8	6	38	25	12	9	24
9	2	2	5	2	8	20	8	5	16	3	15
10	6	8	9	8	9	19	12	26	28	9	22
5	14⋆	12	11	6	15⋆	13	21⋆	40	37	7	20⋆
5	14	0	0	1	15	14	37	0	0	2	34
11	9	4	10	1	13	27	3	11	5	1	41
10	1	5	4	1	13	29	1	9	3	1	43
11	10	16⋆	12	3	20⋆	21	23	41⋆	34	4	56⋆
9	6	18	9	9	20	16	18	53	26	17	40
11	4	2	1	3	11	7	5	2	1	3	10
1	11	4	2	3	10	0	9	4	1	3	9
1	0	8	18⋆	13⋆	19⋆	0	0	9	19⋆	8⋆	22⋆
11	10	9	18	13	19	7	8	8	16	16	30
8	1	3	3	1	9	8	2	1	4	1	16
4	8	1	0	3	10	18	3	0	0	4	22
7	3	9	4	8	13	20	6	15	9	13	38
4	8	1	8	0	10	14	15	9	17	0	35
6	10	11⋆	2	12	14⋆	5	31	15⋆	1	22	33⋆
6	12	13	2	2	15	6	33	26	2	1	35
4	4	11⋆	10	7	14⋆	1	17	17⋆	11	7	21⋆
4	4	4	2	2	10	1	16	14	9	5	19
1	2	2	8⋆	8	11⋆	1	1	3	25⋆	14	13⋆
1	2	2	11	10	11	1	1	4	29	17	22

Résultats pour les mesures DIFF et POSI pour des séries de problèmes comportant 22 activités et 4 ressources.
Le cas où une série de contraintes temporelles généralisées est ajoutée.

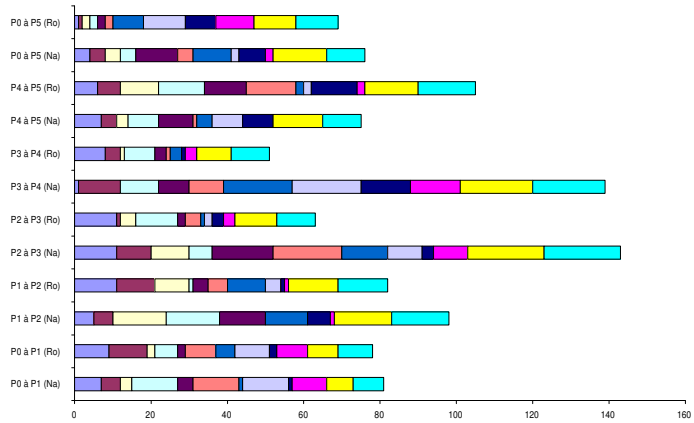


FIG. B.8 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

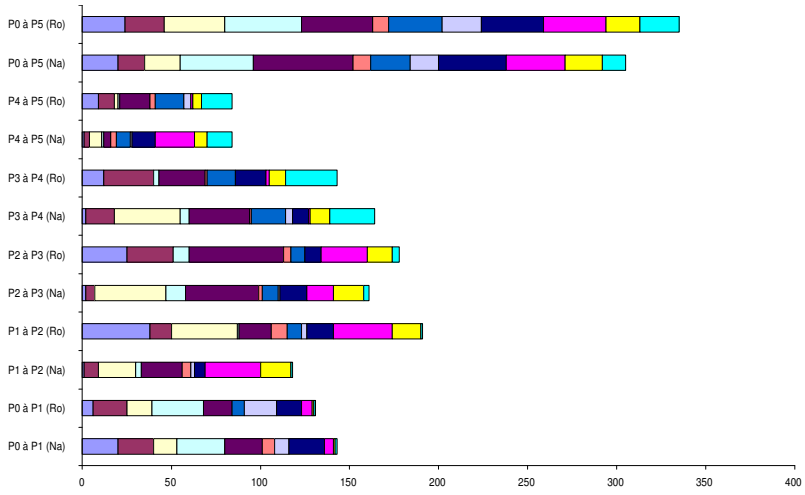


FIG. B.9 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

SDif.						MDif.					
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_4 \to P_5$	$P_0 \to P_5$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_4 \to P_5$	$P_0 \to P_5$
27	3	4	2	1	27	14	1	1	2	1	14
6	65	45	19	20	37	2	26	28	4	6	14
26	7	5	23	2	21	8	4	3	15	1	8
38	19	37	45	20	37	8	5	15	12	4	8
20	64⋆	88	66	13	59⋆	10	13⋆	20	28	4	7⋆
21	86	0	0	2	73	11	28	0	0	2	26
50	9	19	10	1	69	26	1	14	1	1	26
51	1	21	11	1	67	26	1	15	6	1	26
42	41	198⋆	79	10	222⋆	11	11	21⋆	9	4	22⋆
23	22	226	69	34	202	8	10	19	24	8	18
18	4	5	4	3	18	6	1	4	4	1	6
1	18	4	2	3	18	1	6	1	1	1	6
1	0	17	193⋆	37⋆	226⋆	1	0	6	19⋆	5⋆	16⋆
18	17	16	201	37	223	6	6	6	19	4	17
12	2	3	9	1	21	4	2	1	5	1	10
23	9	1	0	3	30	10	2	1	0	1	11
33	6	22	21	27	69	11	4	14	10	11	12
22	18	15	37	0	60	12	9	15	11	0	12
8	58	32⋆	6	39	57⋆	2	13	12⋆	3	10	13⋆
10	54	39	4	3	60	4	11	12	2	2	13
4	27	33⋆	23	17	44⋆	1	9	14⋆	14	5	6⋆
4	22	28	18	6	36	1	9	14	14	5	6
2	6	2	51⋆	29	40⋆	2	4	1	19⋆	14	19⋆
2	6	3	71	35	55	2	4	2	19	7	19

Résultats pour les mesures SDIF et MDIF pour des séries de problèmes comportant 22 activités et 4 ressources.
Le cas où une série de contraintes temporelles généralisées est ajoutée.

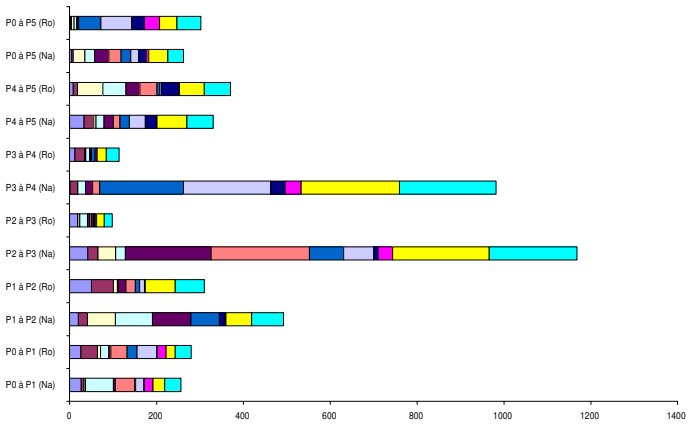


FIG. B.10 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes temporelles généraliséesd'activités est ajoutée.*

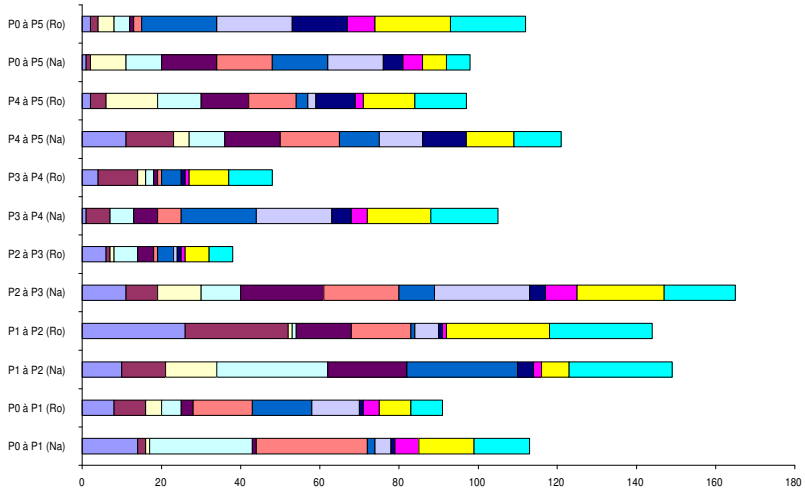


FIG. B.11 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 22 activités et 4 ressources dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

Résultats obtenus pour la stabilté

Ajout d'une série de contraintes temporelles généralisées

Problèmes comportant 32 activités

Diff.						**Posi.**					
$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_4 \to P_5$	$P_0 \to P_5$	$P_0 \to P_1$	$P_1 \to P_2$	$P_2 \to P_3$	$P_3 \to P_4$	$P_4 \to P_5$	$P_0 \to P_5$
1	18	2	2	1	19	1	10	10	6	9	32
18	2	1	2	2	19	11	10	5	10	2	31
2	9	1	3	9	15	14	6	7	1	3	25
2	13	7	3	10	15	14	15	16	1	5	23
8	14	1	9	7	16	24	50	5	26	8	85
5	16	1	2	4	16	17	75	6	4	10	82
2	3	3	1	2	8	21	3	1	2	3	26
3	3	1	4	1	11	19	3	0	2	1	25
5	6	4	7	7	16	10	16	3	25	6	48
4	11	4	16	6	19	9	48	1	64	31	58
9	4	1	1	8	16	7	2	1	1	35	41
10	5	1	1	9	16	11	4	1	1	37	43
1	5	1	5	23⋆	23⋆	18	16	2	8	53⋆	79⋆
1	16	1	0	15	19	18	64	1	0	63	45
16	11	10	1	4	14	33	12	10	1	24	40
5	15	13	4	4	20	34	28	16	16	11	53
7	8	7	3	6	14	5	14	3	4	19	29
5	14	6	2	14	16	9	60	5	2	48	61
8	8	7	7	7	14	11	22	3	16	1	27
11	7	9	7	12	15	18	21	7	17	18	31
6	5	7	4	9	15	7	3	19	1	24	40
4	6	5	9	0	13	3	7	3	47	0	54
10	8	5	1	9	14	60	8	11	2	7	63
9	6	0	1	10	13	58	36	0	2	26	61
5	3	4	7	1	12	19	6	8	17	3	36
7	2	4	7	3	9	40	3	13	27	4	31
8	2	9	1	11	17	13	8	19	3	13	32
12	1	15	0	6	17	14	7	46	0	13	54
3	4	3	13	12	15	7	16	6	32	22	47
3	4	2	15	12	15	7	19	3	51	30	48

Résultats pour les mesures DIFF et POSI pour des séries de problèmes comportant 32 activités.
Le cas où une série de contraintes temporelles généralisées est ajoutée.

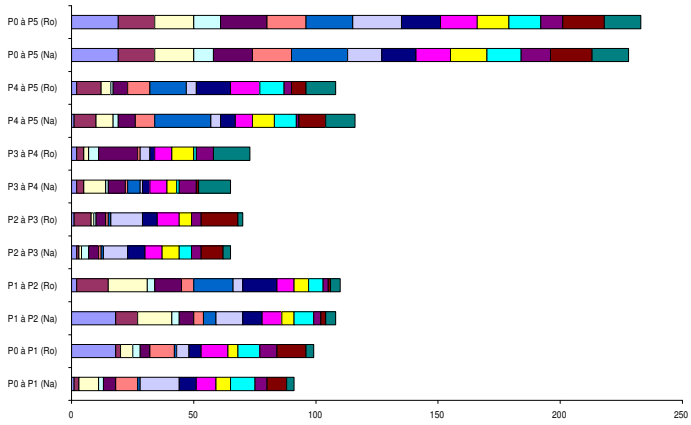


FIG. B.12 – *Évolution des solutions successives pour la mesure de performance* DIFF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes temporelles généraliséesd'activités est ajoutée.*

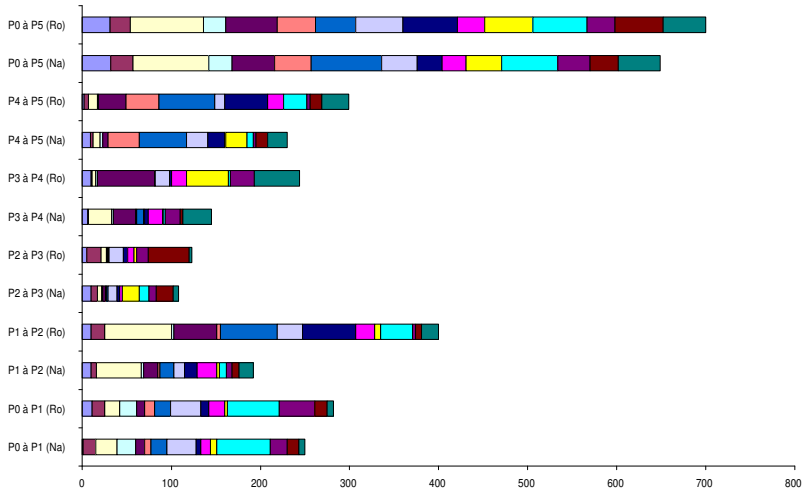


FIG. B.13 – *Évolution des solutions successives pour la mesure de performance* POSI *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

SDif.						MDif.					
$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_0 \rightarrow P_5$	$P_0 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_0 \rightarrow P_5$
1	24	4	3	6	36	1	7	3	2	6	7
25	4	1	8	5	37	7	3	1	6	3	7
26	19	7	3	10	47	23	4	7	1	2	22
26	29	20	3	13	41	23	7	11	1	3	13
29	45	3	30	7	94	17	10	3	18	1	25
21	81	3	7	9	89	17	20	3	5	4	25
25	3	3	2	5	36	17	1	1	2	4	17
26	3	1	7	3	40	17	1	1	2	3	17
9	27	6	30	14	66	3	13	2	7	2	11
8	48	5	81	39	79	3	13	2	15	19	13
9	4	1	1	37	50	1	1	1	1	10	10
13	8	1	1	38	51	4	4	1	1	10	10
13	9	1	6	72$\star$	89$\star$	13	5	1	2	7$\star$	15$\star$
13	59	1	0	76	55	13	16	1	0	18	15
37	14	15	2	16	42	7	2	2	2	9	12
27	37	16	14	8	62	13	4	3	4	3	12
13	26	15	4	23	45	4	7	5	2	7	6
12	93	8	2	59	84	4	13	3	1	11	12
16	38	11	25	10	44	4	13	3	15	2	16
29	32	13	15	27	44	7	13	3	4	8	15
6	5	23	5	34	59	1	1	14	2	13	14
4	6	5	73	0	78	1	1	1	14	0	14
50	8	10	1	9	56	16	1	6	1	1	10
47	22	0	1	25	55	16	17	0	1	11	10
16	11	8	16	1	30	4	5	5	5	1	5
29	9	10	27	3	26	10	5	5	10	1	5
16	9	34	3	25	45	3	8	10	3	5	13
18	10	70	0	27	77	5	10	19	0	13	20
10	20	5	40	32	47	8	11	2	8	5	11
12	21	4	59	36	62	8	11	2	15	8	11

Résultats pour les mesures SDIF et MDIF pour des séries de problèmes comportant 32 activités.
Le cas où une série de contraintes temporelles généralisées est ajoutée.

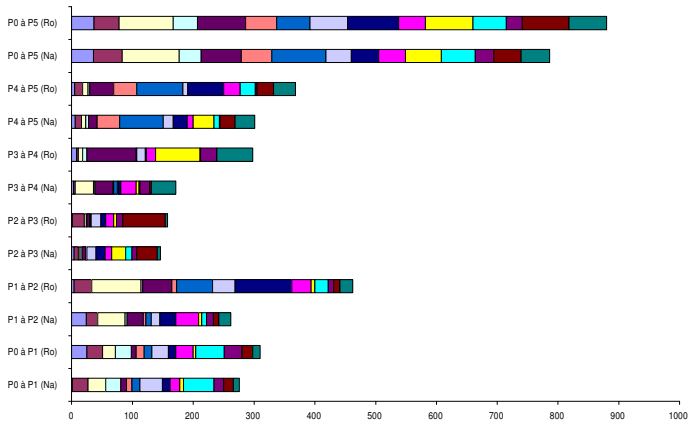


FIG. B.14 – *Évolution des solutions successives pour la mesure de performance* SDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

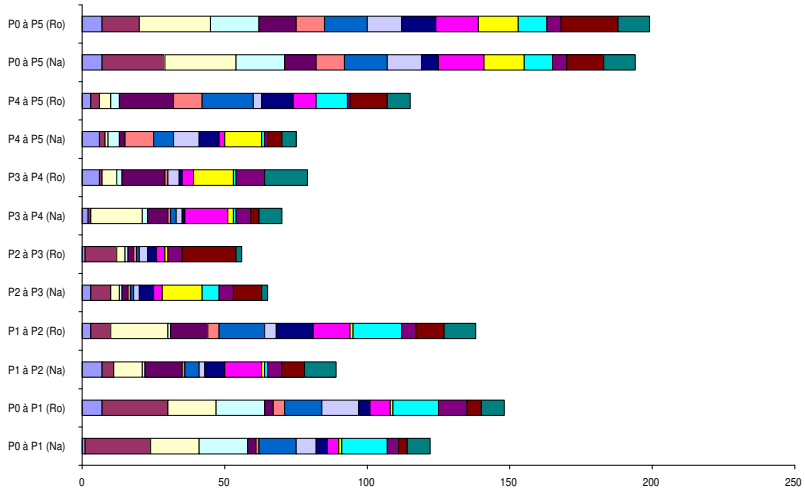


FIG. B.15 – *Évolution des solutions successives pour la mesure de performance* MDIF *sur des séries de problèmes comportant 32 activités dans le cas où une série de contraintes temporelles généralisées est ajoutée.*

Références bibliographiques

[Aggoun et Baldiceanu, 1993] cité pp. 49, 72
A. Aggoun et N. Baldiceanu. Extending chip in order to solve complex scheduling and placement problems. *In Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[Akturk et Gorgulu, 1999] cité pp. 29
M.S. Akturk et E. Gorgulu. Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112:81–97, 1999.

[Aloulou et Portmann, 2002] cité pp. 31
Mohamed Ali Aloulou et Marie-Claude Portmann. A genetic algorithm to achieve scheduling flexibility for a single machine problem. *submitted to RAIRO-Operations Research*, 2002.

[Alvarez-Valdés et Tamrit, 1996] cité pp. 11, 11, 12, 12
R. Alvarez-Valdés et J. M. Tamrit. Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. *R. Slowiński and J. Weglarz, editors, Advances in project scheduling, Elsevier, Amsterdam*, pages 113–134, 1996.

[Artigues *et al.*, 2002] cité pp. 31
C. Artigues, Ph. Michelon, et S. Reusser. Insertion techniques for static and dynamic resource constraint project scheduling. *European Journal of Operational Research*, 2002. à paraître.

[Artigues, 2003] cité pp. 32
C. Artigues. Ordered group assignment of jobs to maximise flexibility in robust scheduling: models and polynomial algorithms. In *International Workshop on Models and Algorithms for Planning and Scheduling Problems MAPSP'03*, Aussois, France, 2003.

[Baar *et al.*, 1998] cité pp. 13
T. Baar, P. Brucker, et S. Knust. Tabu-search algorithms and lower bounds for the resource-contrained project scheduling problem. *Voss S.; S. Martello, I. Osman, and C. Roucirol(Eds): Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer, Dordrecht*, pages 1–18, 1998.

[Baptiste *et al.*, 1999] cité pp. 21
Ph. Baptiste, C. Le Pape, et W. Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.

[Baptiste et Le Pape, 1997] cité pp. 13, 23, 127
Ph. Baptiste et C. Le Pape. Constraint propagation and decomposition techniques for

highly disjunctive and highly cumulative project scheduling problem. In *The Third International Conference on Principales and Practice of Constraint Programming*, numéro 1330 in Lecture Notes in Computer Science. Springer-Verlag, 1997.

[Baptiste, 1998] cité pp. 9, 45
Ph. Baptiste. *Une étude théorique et expérimentale de la propagation des contraintes de ressources*. Thèse de Doctorat, Université de Technologie de Compiégne, 1998.

[Bartusch *et al.*, 1988] cité pp. 22, 22, 22
M. Bartusch, R. H. Möhring, et F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.

[Bean *et al.*, 1991] cité pp. 29
J. Bean, J.R. Birge, J. Mittenthal, et C.E. Noon. Match-up scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.

[Bedworth, 1973] cité pp. 11
D. D. Bedworth. Industrial systems: Planning, analysis, control. *Ronald Press, NewYork*, 1973.

[Beldiceanu et Contejean, 1994] cité pp. 49
N. Beldiceanu et E. Contejean. Introducing global constraint in chip. *Journal of Mathematical and computer Modelling*, 20(12):97–123, 1994.

[Bessière et Cordier, 1993] cité pp. 40
Christian Bessière et Marie-Odile Cordier. Arc-consistency and arc-consistency again. In *AAAI-93: Proceedings 11th National Conference on Artificial Intelligence*, Washington, DC, Juillet 1993. American Association for Artificial Intelligence.

[Bessière, 1991a] cité pp. 44
Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *AAAI-91: Proceedings Ninth National Conference on Artificial Intelligence*, pages 221–226, Anaheim CA, 1991. American Association for Artificial Intelligence.

[Bessière, 1991b] cité pp. 55
Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*, 1991.

[Bierwirth et Mattfeld, 1999] cité pp. 29, 30
C. Bierwirth et D.C. Mattfeld. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7(1):1–17, 1999.

[Blazewicz *et al.*, 1983] cité pp. 7
J. Blazewicz, J.K. Lenstra, et A.H.G. Rinnoy Kan. Scheduling projects subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[Boctor, 1990] cité pp. 12
F. F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 49:3–13, 1990.

[Boizumault *et al.*, 1995] cité pp. 72
P. Boizumault, Y. Delon, C. Guéret, et L. Péridy. Résolution de problèmes en programmation en logique avec contraintes. *Revue d'Intelligence Artificielle - Numéro Spécial: Contraintes*, 9(3):383–406, 1995.

[Boizumault *et al.*, 1996] cité pp. 72
P. Boizumault, Y. Delon, et L. Peridy. Constraint logic programming for examination timetabling. *Journal of Logic Programming*, 26(2):217–233, 1996.

[Brucker *et al.*, 1996] cité pp. 7
P. Brucker, A. Schoo, et O. Thiele. A branch and bound algorithm for the resource-contrained project scheduling problem. Working paper, Department of Mathematics, University of Osnabrück, Osnabrück, Germany, 1996.

[Brucker *et al.*, 1997] cité pp. 16
P. Brucker, J. Hurink, B. Jurisch, et B. Wöstmann. A branch and bound algorithm for the open-shop problem. *Descrete Applied Mathematics*, 76:43–59, 1997.

[Brucker *et al.*, 1998] cité pp. 13, 14, 23, 62, 70, 70, 73
P. Brucker, S. Knust, A. Schoo, et O. Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107(2):272–288, 1998.

[Brucker *et al.*, 1999] cité pp. 6
P. Brucker, A. Drexl, R. Möring, K. Neumann, et E. Pesch. Resource-constrained project scheduling: notation, classification, models and methods. *European Journal of Operational Research*, 1(112):3–41, 1999.

[Brucker et Kampmeyer, 2003] cité pp. 6
P. Brucker et T. Kampmeyer. The rcpsp with time-dependentre source availabilties and its applications. In *International Workshop on Models and Algorithms for Planning and Scheduling Problems MAPSP'03*, Aussois, France, 2003.

[Brucker et Knust, 2000] cité pp. 10, 23
P. Brucker et S. Knust. A linear programming and constraint propagation-based bound for the rcpsp. *European Journal of Operational Research*, 2(127):355–362, 2000.

[Brucker, 1999] cité pp. 6, 6, 13, 14
P. Brucker. Complex scheduling problems. Technical Report D-49069, Department of Mathematics, University of Osnabrück, Osnabrück, Germany, Juin 1999.

[Carlier et Latapie, 1989] cité pp. 15
J. Carlier et B. Latapie. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[Carlier et Latapie, 1991] cité pp. 9, 14
J. Carlier et B. Latapie. Une méthode arborescente pour résoudre les problèmes cumulatifs. *RAIRO Recherche Opérationelle*, 25:311–340, 1991.

[Carlier et Pinson, 1988] cité pp. 14
J. Carlier et E. Pinson. The use of jackson preemptive schedule for solving the job-shop problem. In *International Workshop on Project Management and Scheduling*, Lisbon, 1988.

[Carlier et Pinson, 1994] cité pp. 9
J. Carlier et E. Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 12:269–287, 1994.

[Caseau et Laburthe, 1994] cité pp. 16, 23, 90, 90, 90, 103, 115, 124, 128, 172, 172
Y. Caseau et F. Laburthe. Improved clp scheduling with task-intervals. In *ICLP'94: Proceedings of the Eleventh International Conference on Logic Programming*, Santa Margherita Ligure, 1994.

[Caseau et Laburthe, 1996a] cité pp. 49, 56
Y. Caseau et F. Laburthe. Claire: Combining objects and rules for problem solving. In *in Proceedings of JICSLP'96 workshop on multi-paradigm logic programming*. M.T. Chakravarty, Y. Guo, T. Ida eds. TU berlin, 1996.

[Caseau et Laburthe, 1996b] cité pp. 18, 19, 21, 99
Y. Caseau et F. Laburthe. Cummulative scheduling with task-intervals. In *JICSLP'96: Joint International Conference and Symposium on Logic Programming*, 1996.

[Caseau, 1994] cité pp. 49
Y. Caseau. Constraint satisfaction with an object-oriented knowledge representation language. *Journal of the Applied Intelligence*, 4:157–184, 1994.

[Cheng et Smith, 1994] cité pp. 72
C. Cheng et S. Smith. Generating feasible schedules under complex metric constraints. In *in Proceedings 12th National Conference on Artificial Intelligence*, Août 1994.

[Church et Uzsoy, 1992] cité pp. 29
L.K. Church et R. Uzsoy. Analysis of periodic and event-drive rescheduling policies in dynamic shops. *International Journal Of Computer Integrated Manufacturing*, 5:153–163, 1992.

[Cooper, 1976] cité pp. 12
D. F. Cooper. A note on serial and parallel heuristic for resource-constrained projects: An experimental investigation. *Management Science*, 22:1186–1194, 1976.

[Cooper, 1989] cité pp. 40
Martin Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[Cowling et Johansson, 2002] cité pp. 33
P. Cowling et M. Johansson. Using real time information for effective dynamic scheduling. *European Journal of Operations Research*, 139:230–244, 2002.

[Daniels et Carrillo, 1997] cité pp. 28
R.L. Daniels et J.E. Carrillo. Beta-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29:977–985, 1997.

[Davenport et Beck, 2000] cité pp. 3, 26, 26, 27
A. J. Davenport et J. C. Beck. A survey of techniques for scheduling with uncertainty. *unpublished manuscript*, 2000. available at http://www.eil.utoronto.ca/profiles/chris/chris.papers.html.

[Davies, 1973] cité pp. 11, 11
E. M. Davies. An experimental investigation of resource allocation in mulitactivity projects. *Operation Research Quarterly*, 24:587–591, 1973.

[Davis et Patterson, 1975] cité pp. 12
E. W. Davis et J. H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21:944–955, 1975.

[DeBacker, 1995] cité pp. 71
B. DeBacker. *Méthodes de résolution de disjonctions de contraintes linéaires. Application à la programmation logique avec contraintes.* Thèse de Doctorat, Université d'Orléans, France, 1995.

[Debruyne *et al.*, 2003] cité pp. 44, 55
Romuald Debruyne, Gérard Ferrand, Narendra Jussien, Willy Lesaint, Samir Ouis, et Alexandre Tessier. Correctness of constraint retraction algorithms. In *FLAIRS'03: Sixteenth international Florida Artificial Intelligence Research Society conference*, pages 172–176, St. Augustine, Florida, USA, Mai 2003. AAAI press.

[Debruyne, 1995] cité pp. 44
Romuald Debruyne. Les algorithmes d'arc-consistance dans les csp dynamiques. *Revue d'Intelligence Artificielle – Numéro spécial Contraintes – vol. 1*, 9(3):239–268, 1995.

[Dechter et Dechter, 1988] cité pp. 42
Rina Dechter et Avi Dechter. Belief maintenance in dynamic constraint networks. In *AAAI-88*, pages 37–42, St Paul, MN, 1988.

[Dechter, 1986] cité pp. 48
R. Dechter. Learning while searching in constraint satisfaction problems. In *in Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 178–183, 1986.

[Dechter, 1990] cité pp. 48
R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[Demeulemeester et Herroelen, 1992] cité pp. 7, 14, 23, 115
E. Demeulemeester et W. Herroelen. A branch and bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38(12):1803–1818, 1992.

[Demeulemeester et Herroelen, 1997] cité pp. 117, 128
E. Demeulemeester et W. Herroelen. New benchmark results for the resource-constrained project scheduling problems. *Management Science*, 43:1485–1492, 1997.

[Demmou, 1977] cité pp. 31
R. Demmou. *Etude de familles remarquables d'ordonnancements en vue d'une aide à la décision.* Thèse de Doctorat, Université Paul Sabatier, Toulouse France, 1977.

[Dincbas *et al.*, 1988] cité pp. 72
Mehmet Dincbas, H. Simonis, et P. Van Hentenryck. Solving large scheduling problems in logic programming. In *EURO-TIMS Joint International Conference on Operations Research and Management Science*, Paris, Juillet 1988.

[Dorndorf *et al.*, 1998] cité pp. 22
U. Dorndorf, E. Pesch, et T. Phan Huy. A time-oriented branch-and-bound algorithm for project scheduling with generalized precedence constraints. Working paper, Faculty of Economics, University of Bonn, 1998.

[El Sakkout et Wallace, 2000] cité pp. 45
H. El Sakkout et M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

[Elkhyari *et al.*, 2002a] cité pp. 90, 189
Abdallah Elkhyari, Christelle Guéret, et Narendra Jussien. Conflict-based repair techniques for solving dynamic scheduling problems. In *Principles and Practice of Constraint Programming (CP 2002)*, numéro 2470 in Lecture Notes in Computer Science, pages 702–707, Ithaca, NY, USA, Septembre 2002. Springer-Verlag. Short paper.

[Elkhyari *et al.*, 2002b] cité pp. 70, 73, 189
Abdallah Elkhyari, Christelle Guéret, et Narendra Jussien. Explanation-based repair techniques for solving dynamic scheduling problems. In *AIPS'02 Workshop on On-line Planning and Scheduling*, pages 63–64, Toulouse, France, Avril 2002.

[Elkhyari *et al.*, 2002c] cité pp. 70, 73, 189
Abdallah Elkhyari, Christelle Guéret, et Narendra Jussien. New tools for solving dynamic timetabling problems. In *fourth international conference on the Practice And Theory of Automated Timetabling (PATAT'02)*, pages 112–114, Gent, Belgium, Août 2002.

[Elkhyari *et al.*, 2003a] cité pp. 189, 189, 190
A. Elkhyari, C. Guéret, et N. Jussien. Solving dynamic resource constrained project scheduling problems using new constraint programming tools. In E. K. Burke et P. De Causmaecker, éditeurs, *Practice and Theory of Automated Timetabling IV*, numéro 2740 in Lecture Notes in Computer Science, pages 39–59. Springer-Verlag, 2003.

[Elkhyari *et al.*, 2003b] cité pp. 190
A. Elkhyari, C. Guéret, et N. Jussien. Solving dynamic RCPSP using explanations-based constraint programming. In *MAPSP'03: Sixth workshop on Models and Algorithms for Planning and Scheduling Problems*, pages 119–120, Aussois, France, Mars 2003.

[Elkhyari *et al.*, 2003c] cité pp. 189, 189
A. Elkhyari, C. Guéret, et N. Jussien. Utilisation de la programmation par contraintes pour résoudre le RCPSP dynamique et quelques extensions. In *5ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2003)*, pages 97–98, Avignon, France, Février 2003.

[Elsayed et Nasr, 1986] cité pp. 11
E. A. Elsayed et N. Z. Nasr. Heuristics for resource-constrained project scheduling. *International Journal of Production Research*, 20:95–103, 1986.

[Erschler et Lopez, 1990] cité pp. 19, 90
J. Erschler et P. Lopez. Energy-based approach for task scheduling under time and resource-constraints. In *Second international workshop on Project Management and Scheduling (PMS)*, pages 115–121, Compiègne, France, 1990.

[Erschler et Roubellat, 1989] cité pp. 32
J. Erschler et F. Roubellat. An approach for real time scheduling for activities with time and resource constraints. In *Advances in project scheduling*. R. Slowinski and J. Weglarz Eds, Elsevier, 1989.

[Esswein *et al.*, 2002] cité pp. 31
C. Esswein, C. Artigues, et J.C. Billaut. Maximization of solution flexibility for robust shop scheduling. Technical report, LIA, 2002. report 324.

[Esswein *et al.*, 2003] cité pp. 31
C. Esswein, C. Artigues, et J.C. Billaut. Maximiser la flexibilité sur une machine et dans un job shop : ordonnancements de groupes. In *5ième congrès de la société française de Recherche Opérationnelle et d'Aide à la Décision*, pages 142–143, Avignon, France, 2003.

[flexibilité du Gotha, 2002] cité pp. 26, 33
Groupe flexibilité du Gotha. Flexibilité et robustesse en ordonnancement. *Bulletin de la Roadef*, pages 10–12, Jun 2002. version longue récupérable sur le site : http://www.maths.univ-bpclermont.fr/ sanlavil/FRO.html.

[Fox, 1987] cité pp. 30
M.S. Fox. *Constraint-directed search: a case study of job-shop scheduling*. Morgan-Kaufmann Publishers Inc., 1987.

[Freuder et Wallace, 1992] cité pp. 45
Eugene Freuder et Richard Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[Freuder, 1978] cité pp. 38
Eugene Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21:958–966, Novembre 1978.

[Gaschnig, 1977] cité pp. 48
J. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *In Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 457, Cambridge, MA, 1977.

[Gaschnig, 1979] cité pp. 48, 48
J. Gaschnig. *Performance Measurement and Analysis of Certain search Algorithms*. Thèse de Doctorat, Department of Computer science, Carnegie Mellon University, 1979.

[Ginsberg, 1993] cité pp. 48, 56, 70
M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[Guéret *et al.*, 1998] cité pp. 72
Christelle Guéret, Narendra Jussien, et Christian Prins. Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems. In *PMS'98 International Workshop on Project Management and Scheduling*, Istanbul, Turkey, Juillet 1998.

[Guéret *et al.*, 2000] cité pp. 45
Christelle Guéret, Narendra Jussien, et Christian Prins. Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems. *European Journal of Operational Research*, 127(2):344–354, 2000.

[Haralick et Elliot, 1980] cité pp. 48
R. M. Haralick et G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[Herroelen *et al.*, 1998] cité pp. 6, 22, 22
W. Herroelen, E. Demeulemeester, et B. De Reyck. *A classification scheme for project scheduling*, volume 14, pages 1–26. Weglarz, J. (Ed.): Handbook on Recent Advances in Project Scheduling - recent models, algorithms and applications, International Series in Operations Research and Management Science, Kluwer, 1998.

[Huguet et Lopez, 1999] cité pp. 45
M-J. Huguet et P. Lopez. An integrated constraint-based model for task scheduling and resource assignment. In *Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems CP-AI-OR'99*, Ferrara (Italy), February 1999.

[Jussien *et al.*, 2000] cité pp. 48, 56, 70, 118
Narendra Jussien, Romuald Debruyne, et Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, numéro 1894 in Lecture Notes in Computer Science, pages 249–261, Singapore, Septembre 2000. Springer-Verlag.

[Jussien et Barichard, 2000] cité pp. 49
N. Jussien et V. Barichard. The palm system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, sep 2000.

[Jussien, 1997a] cité pp. 72
N. Jussien. *Relaxation de contraintes pour les problèmes dynamiques.* Thèse de Doctorat, Université de Rennes 1, 1997.

[Jussien, 1997b] cité pp. 45, 52
Narendra Jussien. *Relaxation de Contraintes pour les problèmes dynamiques.* 1. thèse, Université de Rennes I, 24 Octobre 1997.

[Jussien, 2001] cité pp. 43, 44, 55
Narendra Jussien. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus, 1 Décembre 2001.

[Klein et Scholl, 1999] cité pp. 10, 90, 90, 172
R. Klein et A. Scholl. Computing lower bounds by destructive improvement: an application to Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 112:322–345, 1999.

[Klein et Scholl, 2000] cité pp. 23
R. Klein et A. Scholl. Progress: Optimally solving the generalized resource-constrained project scheduling problem. *Mathematical Methods of Operations Research*, 52:467–488, 2000.

[Kolisch *et al.*, 1995] cité pp. 114, 117
R. Kolisch, A. Sprecher, et A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703, 1995.

[Kolisch et Hartmann, 1998] cité pp. 12
R. Kolisch et S. Hartmann. *Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis*, pages 147–178. Weglarz, J. (Ed.): Handbook on Recent Advances in Project Scheduling, Kluwer, Dordrecht, 1998.

[Kolisch, 1996a] cité pp. 11, 12
R. Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14:179–192, 1996.

[Kolisch, 1996b] cité pp. 12
R. Kolisch. Serial and parallel resource-constrained project scheduling methods revised: Theory and computation. *Eurporean Journal of Operational Research*, 90:320–333, 1996.

[Kouvelis et Yu, 1997] cité pp. 33
P. Kouvelis et G. Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publisher, 1997.

[Laburthe, 1998] cité pp. 45
F. Laburthe. *Contraintes et Algorithmes en Optimisation Combinatoire*. Thèse de Doctorat, Université Paris VII-Denis Diderot, Paris, France, 1998.

[Laburthe, 2000] cité pp. 49, 56
F. Laburthe. Choco: implementing a cp kernel. In *TRICS: Post Conference Workshop on Techniques foR Implementing Constraint programming Systems*, Singapour, 2000.

[Lahrichi, 1982] cité pp. 9
A. Lahrichi. Ordonnancements : La notion de parties obligatoires et son application aux problèmes cumulatifs. *RAIRO Recherche Opérationelle*, 16:241–262, 1982.

[Lawrence, 1985] cité pp. 12
S. R. Lawrence. Resource constrained project scheduling - a computational comparison of heuristic scheduling techniques. Rapport technique, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1985.

[Lee et Kim, 1996] cité pp. 13
J.-K. Lee et Y.-D. Kim. Search heuristics for resource constrained project scheduling. *Journal of Operational Research Society*, 47:678–689, 1996.

[Leon *et al.*, 1994] cité pp. 32
V.J. Leon, S.D. Wu, et R.H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[Lhomme, 1993] cité pp. 38
O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. Chambéry, France, 1993.

[Lopez *et al.*, 1992] cité pp. 19, 21
P. Lopez, J. Erschler, et P. Esquirol. Ordonnancement de tâches sous contraintes : une approche énergétique. *RAIRO Automatique, Productique, informatique Industrielle*, 26:453–481, 1992.

[Mauguiére *et al.*, 2002] cité pp. 32
P. Mauguiére, J.-C. Billaut, et C. Artigues. Grouping jobs on a single machine with heads and tails to represent a family of dominant schedules. In *Eighth International Workshop on Project Management and Scheduling PMS 2002*, 2002.

[Mehta et Uzsoy, 1999] cité pp. 27, 28, 33
S. V. Mehta et R. Uzsoy. Predictable scheduling of a single machine subject to breakdowns. *International Journal of Computer Integrated Manufacturing*, 12:15–38, 1999.

[Mingozzi *et al.*, 1998] cité pp. 9
A. Mingozzi, V. Maniezzo, S. Ricciardelli, et L. Bianco. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, 44:714–729, 1998.

[Minton *et al.*, 1992] cité pp. 43
S. Minton, M. Johnston, A. Philips, et P. Laird. Minimizing conflicts: a heuristique repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:160–205, 1992.

[Miyashita, 1995] cité pp. 30
K. Miyashita. Cabins: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence*, 76:377–426, 1995.

[Mohr et Henderson, 1986] cité pp. 40
Roger Mohr et Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(1):225–233, 1986.

[Möhring *et al.*, 1998] cité pp. 22
R. H. Möhring, F. Stork, et M. Uetz. Resource constrained project scheduling with time windows: A branching scheme based on dyna;ic release dates. Rapport technique, Technical University of Berlin, 1998.

[Möhring, 1985] cité pp. 15
R. H. Möhring. *Algorithmic aspects of comparability graphs and interval graphs*, chapter Graphs and Order: The Role of Graphs in the Theory of Ordered Sets and its Applications, NATO Advanced Science Institute Series, Series C., Mathematical and Physical Sciences, pages 41–101. Rival, I. (Ed.), 1985.

[Moukrim *et al.*, 2003] cité pp. 31
A. Moukrim, E. Sanlaville, et R. Guinaud. Parallel machine scheduling with uncertain communication delays. *RAIRO Operations Research*, 37:1–16, 2003.

[Néron, 1999] cité pp. 9
E. Néron. *Du flow-shop hybride au problème cumulatif.* Thèse de Doctorat, Université de Technologie de Compiègne, 1999.

[O'Donovan *et al.*, 1999] cité pp. 28
R. O'Donovan, R. Uzsoy, et K.N. McKay. Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, 37(18):4217–4233, 1999.

[Park *et al.*, 1996] cité pp. 30
J. Park, M. Kang, et K. Lee. Intelligent operations scheduling system in a job shop. *International Journal of Advanced Manufacturing Technology*, 11:111–119, 1996.

[Patterson, 1984] cité pp. 115
J. H. Patterson. A comparison of exact approaches for solving the multiple constrained resource project scheduling problem. *Management science*, 30(7):854–867, 1984.

[Pesant, 2001] cité pp. 54
G. Pesant. A filtering algorithm for the stretch constraint. In *Principles and Practice of Constraint Programming CP'2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 183–195, Paphos, Cyprus, Novembre 2001. Springer-Verlag.

[Picouleau, 2000] cité pp. 30
C. Picouleau. Small perturbations on some up-complete scheduling problems. Research report, CEDRIC, CNAM, Paris, 2000.

[Piechowiak et Rodriguez, 2000] cité pp. 52
Sylvain Piechowiak et J. Rodriguez. Constraint compiling into rules formalism for dynamic csps computing. In *First Workshop on Rule-Based constraint reasoning and programming*, Imperial College, London, UK, Juillet 2000.

[Pinson, 1988] cité pp. 16
E. Pinson. *Le problème de job-shop.* Thèse de Doctorat, Université Paris VI, 1988.

[Prosser, 1993] cité pp. 48
P. Prosser. Domain filtering can degrade intelligent backtracking search. In *in Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 262–267, Chambery, France, 1993.

[Rachamadugu et Morton, 1982] cité pp. 33
R. V. Rachamadugu et T. E. Morton. Myopic heuristics for the single machine weighted tardiness problem. Working paper 30-82-83, GSIA, Carnegie Mellon University, Pittsburgh, 1982.

[Reyck et Herroelen, 1998] cité pp. 22
B. De Reyck et W. Herroelen. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalised precedence constraints. *European Journal of Operational Research*, 111:152–174, 1998.

[Rochart *et al.*, 2003] cité pp. 54
G. Rochart, N. Jussien, et F. Laburthe. Challenging explanations for global constraints.

In *CP03 Workshop on User-Interaction in Constraint Satisfaction (UICS'03)*, Kinsale, Ireland, Septembre 2003.

[Rochart et Jussien, 2003] cité pp. 54
Guillaume Rochart et Narendra Jussien. Une contrainte `stretch` expliquée. In *9ièmes Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'03)*, pages 309–323, Amiens, France, Juin 2003.

[Sabin et Freuder, 1994] cité pp. 48
Daniel Sabin et Eugene Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, éditeur, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, Mai 1994. (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).

[Sabuncuoglu et Bayiz, 2000] cité pp. 28
I. Sabuncuoglu et M. Bayiz. Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research*, 126:567–586, 2000.

[Schäffter, 1997] cité pp. 90
M. Schäffter. Scheduling with respect to forbidden sets. *Discrete Applied Mathematics*, 72:141–154, 1997.

[Schiex et Verfaillie, 1995] cité pp. 45
Thomas Schiex et Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris Mellish, éditeur, *IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence*, Montreal, Août 1995.

[Schwalb et Vila, 1998] cité pp. 45
E. Schwalb et L. Vila. Temporal constraints: A survey. *Constraints*, 3(2/3):129–149, 1998.

[Sevaux et Sörensen, 2002] cité pp. 28, 33
M. Sevaux et K. Sörensen. Genetic algorithm for robust schedules. In *8th International Workshop on Project Management and Scheduling, PMS'02*, 2002. ISBN 84-921190-5-5.

[Smith *et al.*, 1990] cité pp. 30
S.F. Smith, P.S. Ow, J.Y. Potvin, N. Muscotella, et D. Matthys. An integrated framework for generating and revising factory schedules. *Journal of Operational Research Society*, 41(6):539–552, 1990.

[Smith et Cheng, 1993] cité pp. 72
S. Smith et C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Eleventh National Conference on Artificial Intelligencem (AAAI-93)*, pages 139–144, Washington, D.C., 1993.

[Sörensen, 2001] cité pp. 28
K. Sörensen. Tabu seraching for robust solutions. In *4th Mateheuristics International Conference*, pages 707–712, 2001.

[Stinson *et al.*, 1978] cité pp. 7
J. P. Stinson, E. W. David, et B. M. Khamawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 1:252–259, 1978.

[Szelke et Kerr, 1994] cité pp. 30
E. Szelke et R.M. Kerr. Knowledge-based reactive scheduling. *Production planning and control*, 5:124–145, 1994.

[Thomas et Salhi, 1997] cité pp. 12
P. R. Thomas et S. Salhi. An investigation into the relationship of heuristic performance with network-resource characteristics. *Journal of Operational Research Society*, 48:34–43, 1997.

[Ulusoy et Özdamar, 1989] cité pp. 11, 11, 12, 12
G. Ulusoy et L. Özdamar. Heuristic performance and network/resource characteristics in resource-contrained project scheduling. *Journal of Operational Research Society*, 40:1145–1152, 1989.

[Valls *et al.*, 1992] cité pp. 12
V. Valls, M. A. Pérez, et M. S. Quintanilla. Heuristic performance in large resource-constrained projects. Rapport technique, Departement D'Estadistica I Invecigacio Operativa, Universitat de Valencia, 1992.

[Van Hentenryck et Deville, 1991] cité pp. 72
Pascal Van Hentenryck et Yves Deville. The cardinality operator: A new logical connective for constraint logic programming. In Koichi Furukawa, éditeur, *ICLP'91 Proceedings 8th International Conference on Logic Programming*, pages 745–759. MIT Press, 1991.

[Van Hentenryck et Provost, 1991] cité pp. 43
P. Van Hentenryck et L. T. Provost. Incremental search in constraint logic programming. *New Generation Computing*, 9:257–275, 1991.

[Verfaillie et Schiex, 1994] cité pp. 43, 44
Gérard Verfaillie et Thomas Schiex. Dynamic backtracking for dynamic constraint satisfaction problems. In Thomas Schiex et Christian Bessière, éditeurs, *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, Amsterdam, Août 1994.

[Verfaillie et Schiex, 1995] cité pp. 43
Gérard Verfaillie et Thomas Schiex. Maintien de solutions dans les problèmes de satisfaction de contraintes: bilan de quelques approches. *Revue d'Intelligence Artificielle – Numéro spécial Contraintes – vol. 1*, 9(3):269–310, 1995.

[Vieira *et al.*, 2000] cité pp. 29
G.E. Vieira, J.W. Herrmann, et E. Lin. Analytical models to predict the performance of a single-machine system under periodic and event-driven rescheduling strategies. *International Journal of Production Research*, 38(8):1899–1915, 2000.

[Waltz, 1975] cité pp. 38
D. L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. In P. H. Winston, éditeur, *The Psychology of Computer Vision*. McGraw Hill, 1975.

[Wu *et al.*, 1993] cité pp. 29, 30, 33
S. D. Wu, R. H. Storer, et P. C. Chang. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research*, 20:1–14, 1993.

[Wu *et al.*, 1999] cité pp. 31
S. D. Wu, E. Byeon, et R. H. Storer. A graph-theoretic decomposition of the job-shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1):113–124, 1999.

Liste des définitions

Liste des algorithmes

Liste des exemples

Table des figures

Liste des tableaux