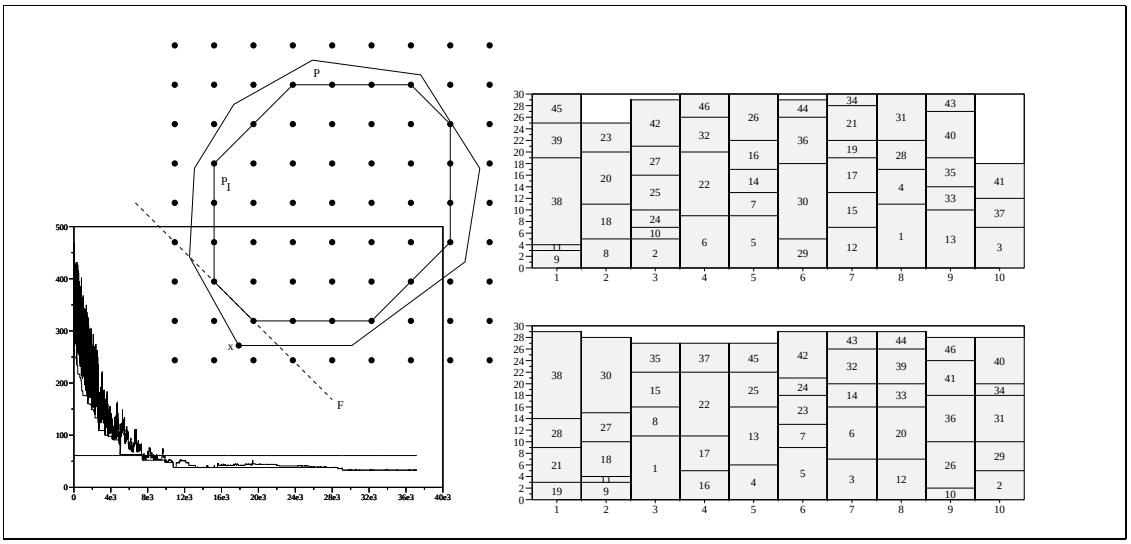
Modèles et algorithmes pour la reconfiguration de systèmes répartis utilisés en téléphonie cellulaire

Renaud Sirdey

Par M. Renaud Sirdey

Modèles et algorithmes pour la reconfiguration de systèmes répartis utilisés en téléphonie cellulaire

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC.



Soutenue le 29 mars 2007
Spécialité : Technologies de l'Information et des Systèmes

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE
UMR CNRS HEUDIASYC

MODÈLES ET ALGORITHMES POUR LA RECONFIGURATION DE SYSTÈMES RÉPARTIS UTILISÉS EN TÉLÉPHONIE CELLULAIRE

THÈSE DE DOCTORAT

pour l'obtention du grade
de docteur de l'Université de Technologie de Compiègne
spécialité Technologies de l'Information et des Systèmes

présentée et soutenue publiquement
par

M. Renaud Sirdey
ingénieur de l'UTC
MSc, Cranfield
architecte système, Nortel

le 29 mars 2007

devant le jury composé de :

M. P. Baptiste	École Polytechnique	rapporteur
M. J.-O. Bouvier	Nortel	examinateur
M. J. Carlier	Univ. de Technologie de Compiègne	directeur de thèse
M. H. Kerivin	Université de Clermont-Ferrand II	examinateur
M. A. R. Mahjoub	Université de Clermont-Ferrand II	rapporteur
M. A. Moukrim	Univ. de Technologie de Compiègne	président
M. D. Nace	Univ. de Technologie de Compiègne	directeur de thèse

À Delphine, Benoît, Clément et Marion,
à mes parents.

Travail récompensé par le prix de thèse « Guy Deniélou » 2008.

Résumé

Ce travail de thèse de doctorat traite de l'étude d'un problème d'ordonnancement NP-difficile au sens fort à contraintes de ressource : le *problème de la programmation des déplacements de processus*. Ce problème, issu de l'industrie des télécommunications, est lié à l'opérabilité de certains systèmes temps réel répartis à haute disponibilité tels le BSCe3, un autocommutateur pour la téléphonie cellulaire commercialisé par Nortel.

En quelques mots, ce problème consiste, étant donnée une répartition arbitraire admissible de processus sur les processeurs d'un système réparti, à trouver une séquence d'opérations (migrations de processus sans effet sur le service ou arrêts temporaires) de moindre impact par le biais de laquelle une autre répartition arbitraire, et fixée à l'avance, peut être obtenue. La principale contrainte réside dans le fait que la capacité des processeurs du système ne doit pas être dépassée durant la reconfiguration.

Nous avons abordé ce problème d'ordonnancement sous différents angles. Tout d'abord, nous avons établi son caractère NP-difficile au sens fort et exhibé quelques cas particuliers polynomiaux. Puis, sur le plan de la résolution exacte dans le cas général, nous avons conçu deux algorithmes de recherche arborescente : le premier trouve ses fondements dans l'étude de la structure combinatoire du problème, le second dans des considérations polyédrales. De nombreux résultats expérimentaux illustrent la pertinence pratique de ces deux algorithmes. Enfin, en raison des contraintes imposées par le caractère temps réel de notre application industrielle, nous avons mis au point un algorithme efficace de résolution approchée basé sur la métaheuristique du recuit simulé et, en capitalisant sur nos travaux en résolution exacte, empiriquement vérifié sa capacité pratique à produire des solutions acceptables, en un sens bien défini.

Abstract

This PhD thesis is devoted to the study of a strongly NP-hard resource-constrained scheduling problem: the *Process Move Programming problem*. This problem arises from the telecommunication industry, in relation to the operability of certain high availability real-time distributed systems such as the BSCe3, a wireless switching system commercialized by Nortel.

Informally, the problem consists, starting from an arbitrary admissible initial distribution of processes on the processors of a distributed system, in finding a least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint lies in the fact that the capacity of the processors must not be exceeded during the reconfiguration.

We have approached this scheduling problem from different angles. First, we have established its strong NP-hardness and exhibited a number of polynomial special cases. Then, in terms of exact resolution in the general case, we have devised two tree search algorithms: one of them is based on the investigation of the combinatorial structure of the problem and the other on polyhedral insights. The practical relevance of these algorithms has been demonstrated through extensive computational experiments. Lastly, motivated by the constraints implied by the real-time nature of our industrial application, we have designed a simulated annealing-based efficient approximate resolution algorithm and, building on our work on exact resolution, empirically demonstrated its practical ability to produce acceptable solutions, in a precisely defined sense.

Remerciements

Ce travail a été réalisé grâce à une collaboration, initiée en mai 2004, entre Nortel, l'UMR CNRS Heudiasyc de l'Université de Technologie de Compiègne et l'Association Nationale de la Recherche Technique.

Je tiens tout d'abord à remercier Messieurs Jacques Carlier, professeur à l'Université de Technologie de Compiègne, et Dritan Nace, maître de conférences à l'Université de Technologie de Compiègne, pour m'avoir fait l'honneur de diriger mes travaux de recherche, pour avoir partagé avec moi leur éminente expérience, tant théorique qu'appliquée, de l'optimisation ainsi que pour avoir réussi à canaliser mon enthousiasme.

Mes remerciements vont ensuite à Monsieur J.-Olivier Bouvier, chef du service d'architecture BSC, Nortel, pour m'avoir encadré au quotidien depuis 2003, pour la confiance qu'il m'a accordée en me suivant dans ce projet un peu fou de mener un travail de thèse de doctorat en sus de mes responsabilités d'architecte système et pour m'avoir servi de modèle de compétence et d'expérience.

Je souhaite également remercier Monsieur Hervé Kerivin, maître de conférences à l'Université de Clermont-Ferrand II, pour une collaboration scientifique des plus fructueuses commencée en janvier 2003 et qui, je l'espère, se poursuivra sur d'autres thèmes.

Aussi, je tiens à remercier Monsieur A. Ridha Mahjoub, professeur à l'Université de Clermont-Ferrand II, pour la bienveillance avec laquelle il a suivi mes travaux sur les polyèdres, pour m'avoir invité à les présenter lors du septième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision ainsi que lors des troisièmes Journées Polyèdres et Optimisation Combinatoire et pour m'avoir fait l'honneur d'accepter de rapporter cette thèse.

J'ai été particulièrement honoré que Monsieur Philippe Baptiste, professeur à l'École Polytechnique, m'invite à présenter mes travaux à l'occasion du séminaire « Algorithmique et optimisation » du laboratoire d'informatique de cette prestigieuse institution et accepte d'être rapporteur de ce manuscrit.

J'adresse aussi mes sincères remerciements à Monsieur Aziz Moukrim, professeur à l'Université de Technologie de Compiègne, pour avoir présidé mon jury de thèse.

Je tiens à exprimer toute ma reconnaissance à Messieurs Djahid Moizaly, ancien leader développement BSC, aujourd'hui leader R&D Accès GSM pour

l'Inde et la Russie, Nortel, et Jean-Michel Duquerrois, leader développement BSC/PCU, Nortel, pour leur soutien tout au long de mon travail de thèse ainsi qu'à Monsieur Daniel Kerivin, chef du service de gestion de projet BSC/PCU, ancien chef du service de vérification BSC, Nortel, pour m'avoir mis en contact, début 2003, avec son frère Hervé et pour sa bienveillance à l'égard de mes travaux.

Merci également à Messieurs Francis Sourd, chargé de recherche à l'Université de Paris VI, et Pierre Fouilhoux, maître de conférences à l'Université de Paris VI, pour m'avoir respectivement invité à présenter certains de mes travaux lors d'un séminaire du Groupe de Recherche en Ordonnancement Théorique et Appliqué ainsi que lors des quinzièmes Journées Franciliennes de Recherche Opérationnelle.

Je suis redevable envers Mesdames Florence Bouthier, Marie-Caroline Huvet et Nathalie Helary de la direction des ressources humaines qui se sont occupées, chez Nortel, du volet administratif de ce projet de thèse ainsi qu'envers Monsieur Jean-Marc Daubeuf, ancien directeur des ressources humaines de Nortel S. A., qui s'est laissé convaincre et qui, par sa signature, a rendu la collaboration entre Nortel, Heudiasyc et l'ANRT, et donc ce travail, possible.

Un immense merci à mon condisciple et ami Nicolas Esposito, enseignant-chercheur à l'Université de Technologie de Compiègne, qui, non content d'avoir pavé la voie en réalisant son travail de thèse de doctorat chez Dassault Systèmes dans des circonstances proches des miennes, a relu l'intégralité de ce mémoire.

Merci aussi à Messieurs Laurent Dorys-Charnalet et Pierre-Marie Trancrez, tous deux architectes système BSC, Nortel, qui, avec J.-Olivier Bouvier et Daniel Kerivin, ont assisté à l'ultime répétition de ma soutenance.

Je dois également exprimer tout mon amour à mon épouse, Delphine, qui m'a susurré, à ses risques et périls, l'idée de me lancer dans un travail de thèse au lieu de prendre un stagiaire de DEA, ainsi qu'à mes trois p'tits loups Benoît, né le 27 mars 2004, Clément, né le 31 octobre 2006, et Marion, née le 14 août 2009. J'espère avoir su rester raisonnable au cours de ces trois années et ne pas avoir passé plus de temps que de rigueur à découper des « tout petits topes » (*dixit* Benoît).

Enfin, tout ceci serait bien incomplet sans un hommage à la mémoire de mon père, le contre-amiral Christian Sirdey, brutalement emporté par un cancer le 13 novembre 2006, ainsi qu'un témoignage d'affection à ma mère, Annick, et à mon frère, Mathieu.

Table des matières

Introduction

L'optimisation combinatoire, domaine qui traite des problèmes où il convient de trouver un meilleur élément parmi un ensemble de taille finie mais astronomique, est l'une des branches les plus appliquées des mathématiques.

Historiquement, l'industrie des télécommunications a toujours été l'un des principaux pourvoyeurs des problèmes combinatoires les plus redoutables, principalement dans les domaines liés à la conception, au dimensionnement et à l'exploitation des grands réseaux. La conception des équipements en charge de l'écoulement du trafic au cœur de ces réseaux, les *autocommutateurs*, se trouve aussi être à la source de problèmes combinatoires aussi divers que variés.

Brièvement, un autocommutateur est un système qui fournit une quantité limitée de certaines ressources (circuits, ports, liens, CPU, mémoire, etc.) interdépendantes qu'il convient d'utiliser efficacement. Dans un tel contexte, l'omniprésence de problématiques combinatoires n'est en rien surprenante. De plus, le caractère temps réel embarqué de ces systèmes impose un nombre important de contraintes, par exemple sur le temps d'exécution ou sur la taille du logiciel à embarquer, quant aux solutions algorithmiques utilisables, en particulier lorsqu'il s'agit de résoudre des problèmes *intrinsèquement* difficiles (NP-difficiles, en termes techniques).

C'est dans ce contexte que s'inscrivent nos travaux.

Ce mémoire traite donc de l'étude d'un problème d'ordonnancement NP-difficile au sens fort à contraintes de ressource, le *problème de la programmation des déplacements de processus* ou, pour alléger, *problème de reconfiguration*. Ce problème est lié à l'opérabilité de certains systèmes répartis temps réel à haute disponibilité, tels le BSCe3, BSC[1] de dernière génération commercialisé par Nortel.

En quelques mots, étant donné un système réparti composé de processeurs sur lesquels tournent des processus, notre problème consiste à ordonner des déplacements de processus de telle manière que des contraintes de capacité sur les processeurs du système ne soient jamais violées. Les situations de blocage sont résolues en arrêtant temporairement des processus donc en renonçant temporairement à rendre une partie du service, l'objectif étant naturellement d'avoir le

1. Un BSC (*Base Station Controller*) est un autocommutateur en charge de la gestion de la ressource radio et du premier niveau de concentration du trafic dans un réseau GSM.

moins possible recours à ce mécanisme.

Dans le contexte du BSCe3, ce problème se présente lorsqu'il s'agit de restaurer, sans arrêt du système, des propriétés (par exemple l'équirépartition de la charge de traitement d'appels) qui se perdent au gré des inévitables défaillances de processeurs et des modifications dynamiques de l'ensemble des équipements (principalement des stations de base) gérés par le système. D'autres motivations telles que l'obtention de répartitions de processus favorables à certaines stratégies de mises à jour logicielle sans arrêt du système méritent également d'être mentionnées.

Ce manuscrit comporte trois parties.

La première partie a pour ambition de fournir une introduction à l'optimisation combinatoire qui soit accessible à l'industriel non spécialiste ainsi qu'une introduction à l'architecture des réseaux et systèmes sur lesquels nous travaillons qui soit, elle, accessible à l'universitaire non spécialiste.

La deuxième partie présente notre contribution de manière succincte. En particulier, nos principaux résultats y sont énoncés sans preuve.

Enfin, la troisième partie de ce mémoire s'organise sous la forme d'un recueil regroupant nos principales publications soumises ou acceptées dans des journaux spécialisés de la littérature anglo-saxonne. Cette partie présente donc notre contribution de manière approfondie tout en autorisant plusieurs lectures, les articles pouvant être lus indépendamment. Précisons toutefois que cette partie se conclut par un article légèrement hors propos mais qu'il nous a semblé opportun d'inclure : il illustre bien les problèmes combinatoires que nous avons régulièrement à traiter dans le cadre de nos responsabilités opérationnelles d'architecte système.

Nous résumons ci-après notre contribution à l'étude du problème de reconfiguration.

Commençons par formuler le problème de manière un peu plus précise. Soit l'ensemble des processus et l'ensemble des processeurs d'un système réparti. Un état du système consiste en une distribution d'un sous-ensemble des processus sur les processeurs de manière à ce que les ressources consommées sur ces derniers n'excèdent pas leur capacité. Un processus peut être déplacé d'un processeur vers un autre de deux manières : soit il est *migré*, auquel cas il consomme des ressources sur les processeurs source et cible du déplacement pour la durée de la migration, soit il est *interrompu*, c'est-à-dire arrêté puis ultérieurement redémarré sur le processeur cible du déplacement. Soit M l'ensemble des déplacements nécessaires afin de faire passer le système d'un état arbitraire, l'*état initial*, vers un autre, l'*état final*, notre problème consiste à trouver $I \subset M$ et σ un ordonnancement réalisable des déplacements de $M \setminus I$ tel que $\sum_{m \in I} c_m$ soit minimum, c_m étant le coût d'interrompre le processus associé à m et I l'ensemble des déplacements interrompus (les interruptions étant toutes réalisées au début).

L'un des premiers résultats que nous avons obtenu concerne la complexité du problème de reconfiguration : nous avons montré qu'il est NP-difficile au sens fort. Ce résultat est important dans la mesure où il permet d'orienter notre

recherche vers certaines techniques algorithmiques plutôt que vers d'autres : à moins que la célèbre conjecture $P \neq NP$ soit fausse, il n'existe pas d'algorithme *efficace* (*polynomial*[2] voire *pseudopolynomial*) capable de résoudre exactement le problème, c'est-à-dire qui trouve toujours une solution de valeur minimum. Ceci justifie les approches suivantes :

1. chercher s'il existe des cas particuliers du problème qu'il est possible de résoudre exactement à l'aide d'algorithmes efficaces ;
2. chercher à résoudre le problème de manière exacte à l'aide d'algorithmes non polynomiaux ;
3. chercher à résoudre le problème de manière approchée à l'aide d'algorithmes efficaces.

Les contraintes sur le temps d'exécution imposées par le contexte temps réel de notre application font que c'est l'approche 3 ci-dessus qui se prête le mieux à l'obtention de méthodes de résolution viables sur le plan industriel. Les autres approches sont néanmoins complémentaires : en particulier, l'approche 2 va permettre d'étudier la qualité des solutions obtenues à l'aide des méthodes issues de l'approche 3.

Nous avons voulu réaliser un travail éclectique, nous avons donc attaqué le problème de reconfiguration sous l'angle de ces trois approches.

Nos travaux sur les cas particuliers polynomiaux se basent sur la notion de *multigraphe de transfert* que nous avons introduite. Le multigraphe de transfert est défini comme le multigraphe[3] orienté dont les sommets sont les processeurs et les arcs les déplacements de processus.

Lorsque le multigraphe de transfert ne contient pas de circuit, nous avons montré que les déplacements peuvent toujours être ordonnés de manière à ce qu'aucune interruption ne soit nécessaire. Un tel ordonnancement se déduit aisément d'une *bonne numérotation* des sommets du graphe, c'est-à-dire d'une fonction f de l'ensemble des sommets vers l'ensemble des entiers qui vérifie $f(v) < f(w)$ pour tout arc $\{v, w\}$ appartenant au graphe.

De plus, lorsque le multigraphe de transfert contient des circuits, nous avons montré qu'il convient de traiter ses composantes fortement connexes indépendamment et selon un ordre qui se déduit d'une bonne numérotation des sommets du graphe de transfert réduit (graphe obtenu en contractant les composantes fortement connexes). Ceci nous donne une première méthode de décomposition.

Par ailleurs, lorsque les processeurs n'offrent qu'un seul type de ressources et lorsque tous les processus ont la même consommation, nous avons montré que le problème de reconfiguration peut être résolu de manière efficace. Pour ce

2. C'est-à-dire tel que le nombre d'opérations élémentaires effectuées lors d'une exécution de l'algorithme est borné par un polynôme de la taille du problème (ici, le nombre de déplacements).

3. Les arcs parallèles sont autorisés, les boucles ne le sont pas.

faire, nous avons proposé un algorithme, démontré sa validité et prouvé que sa complexité est bien polynomiale.

Sur le plan de la résolution exacte dans le cas général, nous avons d'abord approché le problème de manière combinatoire, c'est-à-dire en exploitant directement sa structure dans le cadre d'un algorithme d'énumération par séparation et évaluation (*branch-and-bound*).

Les algorithmes d'énumération par séparation et évaluation sont des algorithmes de recherche arborescente dont le succès est basé sur la notion d'énumération implicite : des pans entiers de l'arborescence sont élagués par exemple lorsque l'évaluation par défaut d'une solution partielle indique qu'il est impossible de la compléter de manière à fournir une solution de valeur moindre que la meilleure solution déjà rencontrée. Les performances pratiques d'un tel algorithme dépendent crucialement de la qualité de l'évaluation par défaut mais aussi de la présence de *règles de dominance*, règles qui permettent d'identifier et donc d'élaguer des branches redondantes de l'arborescence.

Au lieu de chercher, à partir de l'état initial, à atteindre l'état final, il s'avère plus fructueux, dans le cadre d'un algorithme de recherche arborescente, de chercher à améliorer la pire des solutions (celle qui consiste à interrompre tous les déplacements), en évitant certaines interruptions. Selon ce point de vue, une solution partielle est représentée par un triplet composé d'un ensemble d'interruptions définitives, d'un ensemble d'interruptions non définitives et d'un ordonnancement admissible des déplacements non interrompus. Typiquement, l'interruption d'un déplacement peut être évitée si la capacité résiduelle du processeur source de ce déplacement est toujours supérieure à sa consommation durant l'exécution de l'ordonnancement des déplacements non interrompus. Si c'est le cas alors le processus déplacé peut rester sur le processeur source durant l'exécution de l'ordonnancement et, ensuite, être déplacé vers le processeur cible.

Cette manière de voir le problème nous a permis d'obtenir des bornes inférieures, la plus intéressante d'entre elles nécessitant la résolution d'un problème de sac à dos par processeur. Nous avons utilisé cette borne dans le cadre d'un algorithme de recherche arborescente exploitant aussi une règle de dominance assez générale. Cette dernière permet d'éliminer plusieurs sources importantes de redondance. L'une d'entre elles, par exemple, concerne les ordonnancements totaux équivalents car ils induisent les mêmes ordonnancements restreints pour chacun des processeurs.

Nous avons illustré la pertinence pratique de cet algorithme en l'utilisant pour attaquer des instances *difficiles*[4]. En particulier, nous avons pu résoudre exactement des instances ayant jusqu'à 190 déplacements en moins de vingt minutes. En pratique néanmoins, si le temps de calcul est limité à vingt minutes, l'algorithme n'a de bonnes chances de succès que jusqu'à de l'ordre de 50 déplacements.

Toujours sur le plan de la résolution exacte dans le cas général, nous avons en-

4. À la fois en termes de taille mais aussi de capacité résiduelle des processeurs.

suite abordé le problème de manière géométrique, à l'aide de l'approche polyédrale.

Cette approche trouve ses fondements dans une interprétation géométrique des problèmes combinatoires : lorsque la fonction économique est linéaire, c'est l'étude du polytope (un polytope est un polyèdre borné) correspondant à l'enveloppe convexe des vecteurs caractéristiques des solutions du problème qui guide la conception de l'algorithme de résolution. En particulier, les inégalités nécessaires à la description de ce polytope (on les appelle des *facettes*) sont extrêmement pertinentes quant à la mise en œuvre d'algorithmes de résolution exacte performants [5].

L'application des méthodes polyédriques à notre problème a tout d'abord requis l'étude théorique de deux polytopes que nous avons introduits : le polytope des sous-tournois sans-circuit et le polytope des programmes de déplacements, le premier étant une relaxation du second. Nous avons, en particulier, étudié les propriétés élémentaires de ces polytopes (dimension et facettes élémentaires, notamment), exhibé plusieurs classes d'inégalités qui en définissent des facettes et étudié les problèmes de séparation associés. L'idée était de commencer par l'étude du polytope des sous-tournois sans-circuit, d'un abord plus direct que celui des programmes de déplacements, de manière à faciliter l'étude théorique de ce dernier. Cette approche s'est avérée fructueuse : nous avons pu montrer que les classes de facettes que nous avons identifiées pour le polytope des sous-tournois sans circuit définissent aussi des facettes du polytope des programmes de déplacements sous des hypothèses peu restrictives. Aussi, nous avons exhibé deux classes d'inégalités, les contraintes dites de s- et de t-recouvrement, qui définissent des facettes du polytope des programmes de déplacements, sous des conditions peu restrictives, et qui sont séparables en temps pseudopolynomial par résolution de quelques problèmes de sac à dos.

Le passage à la pratique s'est fait par le biais d'un algorithme de recherche arborescente polyédrique (*branch-and-cut*) dont le principal ingrédient est une relaxation linéaire comprenant un nombre exponentiel de contraintes qui définissent des facettes du polytope des programmes de déplacements. Cette relaxation, dont on peut théoriquement venir à bout en temps pseudopolynomial à l'aide de l'algorithme de l'ellipsoïde, est résolue à chaque nœud de l'arbre de recherche par un algorithme de coupe.

Là encore, nous avons illustré la pertinence pratique de cet algorithme en l'utilisant pour attaquer des instances *difficiles*[4]. En particulier, nous avons pu résoudre exactement des instances ayant jusqu'à 119 déplacements (pour un système à 70 processeurs) en moins de quatre heures. En pratique cependant, si le temps de calcul est limité à quatre heures, l'algorithme n'a de bonnes chances de succès que jusqu'à de l'ordre de 80 déplacements. Lorsque la taille des instances augmente, l'algorithme a néanmoins été en mesure de fournir des solutions approchées de bonne qualité c'est-à-dire, ici, *prouvées* ne se situer qu'à moins de

5. Bien que non efficaces, au sens théorique du terme.

5% (généralement moins) de l'optimum, à la quasi-totalité des instances (de taille allant jusqu'à 180 déplacements) sur lesquelles nous l'avons essayé, toujours sous la contrainte d'une limitation du temps de calcul à quatre heures.

Notons aussi que nous avons utilisé l'algorithme de recuit simulé discuté ci-après pour obtenir une bonne solution initiale, ce qui permet un premier élagage significatif de l'arborescence de recherche. Ceci illustre la complémentarité pratique entre une méthode de résolution approchée soignée et un algorithme polyédrique, complémentarité qui s'avère d'autant plus pertinente que les instances sont de grande taille.

Sur le plan de la résolution approchée, enfin, nous avons donc abordé le problème à l'aide de la méthode du recuit simulé.

La méthode du recuit simulé a été proposée dans les années 80 et, à l'origine, justifiée par analogie avec la technique dite du recuit utilisée par les physiciens afin de conduire certains systèmes physiques vers un état de basse énergie. Il s'agit d'une *métaheuristique*, autrement dit d'un principe général de construction d'algorithmes de résolution approchée pour les problèmes combinatoires difficiles. Les principaux avantages de cette méthode sont d'être relativement bien comprise sur le plan théorique et de conduire à des algorithmes assez simples.

Sur le plan théorique nous avons introduit la notion de solution (α, β)-acceptable, où β définit une exigence de qualité (distance à l'optimum) et où α est la probabilité de satisfaire cette exigence. Nous avons ensuite utilisé la théorie des chaînes de Markov, sous-jacente à la méthode du recuit simulé, afin de dégager des conditions sous lesquelles des solutions (α, β)-acceptables peuvent être obtenues. Enfin, nous avons appliqué ces résultats de manière à obtenir un algorithme pseudopolynomial de résolution approchée pour notre problème.

Bien que notre raisonnement ne soit au final qu'heuristique, car il nous faudrait fournir une caractérisation pratique de la vitesse de convergence de certaines chaînes de Markov vers leur loi stationnaire [6], nous l'avons vérifié empiriquement.

À cet effet, nous avons paramétré notre algorithme de manière à ce qu'il produise des solutions $(95\%, 5\%)$-acceptables, c'est-à-dire des solutions situées à moins de 5% de l'optimum 95 fois sur 100, et, grâce aux résultats obtenus à l'aide de nos méthodes de résolution exacte, montré que l'algorithme a effectivement fourni des solutions 5%-acceptables à 97.74% des 1020 instances *difficiles*[4] sur lesquelles nous l'avons essayé. Ceci, ainsi que l'analyse du nombre d'essais nécessaires à l'obtention d'une solution 5%-acceptable, suggère que l'algorithme est bien, en pratique, capable de produire des solutions $(95\%, 5\%)$-acceptables (avec une probabilité comprise entre 94.08% et 96.76%).

Aussi, précisons que ce dernier algorithme est tout à fait viable sur le plan de notre contexte industriel, tant en termes de qualité des solutions produites et de temps d'exécution (les instances réelles s'avérant beaucoup plus faciles que nos instances de test, car de fait moins contraintes) qu'en termes de complexité et de

6. Il s'agit là d'un problème extrêmement difficile.

maintenabilité du logiciel résultant. Pour se fixer les idées, cet algorithme, dont le cœur ne représente qu'une petite centaine de lignes de programmation, a pu résoudre à l'optimum toutes les instances réelles que nous lui avons présentée en moins d'une dizaine de secondes.

Liste de publications

Journaux

- R. Sirdey, J. Carlier and D. Nace, « *A GRASP for a resource-constrained scheduling problem* », International Journal of Innovative Computing and Applications 2:143-149 (numéro spécial *Metaheuristics and Real-World Problems*), 2010 ;
- R. Sirdey, J. Carlier and D. Nace, « *Approximate resolution of a resource-constrained scheduling problem* », Journal of Heuristics 15:1-17, 2009 ;
- R.Sirdey, « *Models and algorithms for the reconfiguration of distributed wireless switching systems* », 4OR 6:195-198, 2008 ;
- R. Sirdey, « *Combinatorial optimization problems in wireless switch design* », 4OR 5:319-333, 2007 ;
- R. Sirdey and H. Kerivin, « *A branch-and-cut algorithm for a resource-constrained scheduling problem* », RAIRO Operations Research 41:235-251 (numéro spécial *Polyhedra and Combinatorial Optimization*), 2007 ;
- R.Sirdey, J. Carlier, H. Kerivin and D. Nace, « *On a resource-constrained scheduling problem with application to distributed systems reconfiguration* », European Journal of Operational Research 183:546-563, 2007.

Rapports de recherche

- R. Sirdey and H. Kerivin, « *Polyhedral combinatorics of a resource-constrained ordering problem part I—On the partial linear ordering polytope* », rapport technique Nortel GSM Access R&D PE/BSC/INF/017912 V01/EN, 2006 ;
- H. Kerivin and R. Sirdey, « *Polyhedral combinatorics of a resource-constrained ordering problem part II—On the process move program polytope* », Technical report Nortel GSM Access R&D PE/BSC/INF/017913 V01/EN, 2006.

Communications

- R. Sirdey, « Optimisation combinatoire et autocommutateurs », Séminaire algorithmique et optimisation, École Polytechnique, septembre 2006 ;
- R. Sirdey, « Problèmes d'optimisation combinatoire en conception d'autocommutateurs », JFRO, Université Paris VI, juin 2006 ;
- R. Sirdey, « Polyèdres et reconfiguration dynamique d'autocommutateurs répartis », séminaire du GOTHA, Université Paris VI, mai 2006 ;
- R. Sirdey and H. Kerivin, « Sur le polytope des sous-tournois sans circuit », ROADÉF, Lille, février 2006 ;
- R. Sirdey and H. Kerivin, « Polyèdres et reconfiguration dynamique d'autocommutateurs répartis », JPOC3, Avignon, juin 2006 ;
- H. Kerivin and R. Sirdey, « Polyhedral Combinatorics of Some Capacity-constrained Ordering Problems », INFORMS, San-Francisco, novembre 2005.
- R. Sirdey, « Problèmes d'optimisation combinatoire et autocommutateurs utilisés en téléphonie mobile », séminaire du LIMOS, Université de Clermont-Ferrand II, mars 2005.

Vulgarisation

- R. Sirdey, « Optimiser... En découpant des polyèdres », L'Ouvert 115 :51-61, 2007 ;
- R. Sirdey, « Des solutions pour faire bonne figure », La Recherche 407 :82-83, 2007 ;
- R. Sirdey, « Sudokus et programmation linéaire », Quadrature 63 :9-13, 2007 ;
- R. Sirdey, « Sudokus et algorithmes de recuit », Quadrature 62 :7-10, 2006.

Première partie

Préliminaires

Introduction

Cette première partie introduit les éléments nécessaires à une bonne compréhension des aspects tant mathématiques qu'industriels de nos travaux.

Le chapitre 1 fournit quelques rudiments d'optimisation combinatoire. En particulier, nous nous sommes donné pour objectif de répondre aux questions : qu'est-ce qu'un problème combinatoire ? Qu'est-ce qu'un algorithme efficace ? Qu'est-ce qu'un problème *intrinsèquement* difficile ? Nous passons ensuite en revue les principales stratégies de résolution des problèmes combinatoires intrinsèquement difficiles. En sus, nous insistons sur l'approche polyédrale, une approche géométrique qui conduit à des méthodes dont la pertinence pratique n'est plus à démontrer lorsqu'il s'agit de résoudre de manière exacte ou approchée (avec une mesure fine de l'écart à l'optimum) des problèmes combinatoires difficiles.

Le chapitre 2, quant à lui, est dédié au contexte industriel dans lequel s'inscrivent nos travaux : il s'agit d'un problème de reconfiguration d'un BSC, autocommutateur qui s'insère dans un réseau GSM. Nous répondons donc aux questions : qu'est-ce qu'un réseau GSM ? Qu'est-ce qu'un BSC ? Nous présentons ensuite le BSCe3, le BSC de dernière génération commercialisé par Nortel, son architecture matérielle ainsi que ses mécanismes de tolérance aux pannes. Si l'ingénieur Nortel peut vraisemblablement sauter ce chapitre, le lecteur universitaire y trouvera de quoi se faire une idée sur un contexte industriel pourvoyeur de problèmes combinatoires [7].

Précisons, enfin, que le lecteur uniquement intéressé par les aspects mathématiques de nos travaux peut faire l'économie de cette partie et se reporter directement à la partie II.

7. Ainsi qu'illustré dans Sirdey (2006a).

Chapitre 1

Introduction à l'optimisation combinatoire

Introduction

L'optimisation combinatoire est la branche des mathématiques qui traite des problèmes où il convient de trouver un meilleur élément, au sens d'une fonction économique, parmi un ensemble de taille finie mais astronomique, ceci sans avoir recours à l'examen, impossible en pratique, de tous les éléments de l'ensemble.

Le problème bien connu du voyageur de commerce est emblématique. Il s'agit, étant donné un ensemble de villes et, pour chaque couple de villes, la distance qui les sépare, de trouver une tournée, c'est-à-dire un circuit passant une et une seule fois par chacune des villes, de longueur minimale. S'il y a n villes alors il y a $n!$ tournées possibles et, typiquement, $23! = 25\ 852\ 016\ 738\ 884\ 976\ 640\ 000$ est déjà du même ordre de grandeur que le nombre de microsecondes qui se sont écoulées depuis le *big bang* !

Un grand nombre de problèmes d'optimisation combinatoire sont intrinsèquement difficiles (c'est le cas de l'exemple ci-dessus), ce qui signifie qu'il est fort vraisemblable qu'il n'existe pas d'algorithme permettant de les résoudre qui soit efficace, autrement dit qui soit capable de faire significativement mieux, *dans le pire des cas*, que l'examen exhaustif du nombre exponentiel de solutions.

Ces résultats négatifs ne doivent pas obscurcir le tableau outre mesure : pour un problème intrinsèquement difficile donné, il existe généralement des familles d'instances particulières qui peuvent, elles, être résolues efficacement et, surtout, il peut s'avérer que la majeure partie des instances rencontrées en pratique ne soient pas si difficiles.

Ce chapitre a pour ambition de fournir une introduction à l'optimisation combinatoire qui soit accessible au non spécialiste (il peut donc vraisemblablement être sauté par le lecteur averti) : après quelques rappels sur la théorie de la complexité, nous passons en revue les différentes stratégies algorithmiques d'ap-

proche des problèmes combinatoires difficiles. Enfin, dans une section un peu plus technique, nous exposons quelques éléments de la théorie des polyèdres, théorie dans laquelle l'approche polyédrale, que nous présentons brièvement, trouve ses fondements.

1.1 Rappels sur la théorie de la complexité

Cette section vise à rappeler, de manière relativement informelle, quelques éléments de base de la théorie de la complexité. Pour plus de détails, nous nous permettons de renvoyer le lecteur à la littérature spécialisée, en particulier à l'ouvrage bien connu de Garey & Johnson (1979).

1.1.1 La complexité des algorithmes

La complexité d'un algorithme est une fonction de la taille de l'instance en entrée (par exemple le nombre de villes dans le cas d'une instance du problème du voyageur de commerce) qui fournit, à un facteur multiplicatif près, une borne supérieure sur le temps d'exécution de l'algorithme. Dans la mesure où cette borne est indépendante de l'instance, on parle de complexité au pire.

Plus précisément, la complexité d'un algorithme de résolution d'un problème donné est en $O(f(n))$ s'il existe une constante $\alpha > 0$ telle que l'algorithme s'arrête après au plus $\alpha f(n)$ opérations élémentaires (addition, multiplication, accès à un tableau, déréférencement d'un pointeur, etc.) lorsque l'on lui présente une instance de taille n.

1.1.2 Notion d'algorithme efficace

On dit qu'un algorithme est *polynomial* ou tout simplement *efficace*, si sa complexité est un polynôme, autrement dit, s'il existe une constante $k > 0$ tel que l'algorithme s'arrête au plus après $O(n^k)$ opérations élémentaires.

Identifier les algorithmes efficaces aux algorithmes polynomiaux est assez naturel. Par exemple, pour un problème linéaire (polynomial de degré 1), doubler la vitesse de l'ordinateur utilisé pour le résoudre revient *grosso modo* à doubler la taille des instances que l'on peut résoudre dans un intervalle de temps donné. Pour un problème dont la complexité est exponentielle, par contre, doubler la vitesse de l'ordinateur permet seulement d'ajouter une constante à la taille des instances que l'on peut traiter dans le même intervalle de temps !

Il existe de nombreux problèmes d'optimisation combinatoire qui peuvent être résolus efficacement, on dit aussi *en temps polynomial*. Par exemple, le problème du plus court chemin, les problèmes de flots (Ahuja et al., 1993) et les problèmes de couplages (Lovász & Plummer, 1986) sont tous des problèmes polynomiaux.

1.1.3 Les problèmes *NP*-difficiles

Il existe aussi de nombreux problèmes d'optimisation combinatoire que l'on ne sait pas, aujourd'hui, résoudre efficacement, il s'agit des problèmes NP-difficiles.

L'ensemble NP, introduit par Cook (1971), est l'ensemble des problèmes de décisions, c'est-à-dire des questions dont la réponse est soit oui soit non, tels que, pour chacune des instances dont la réponse est oui, il existe un certificat qui permet de montrer en temps polynomial que la réponse est bien oui. Par exemple, si la question est de savoir, pour une instance du problème du voyageur de commerce, s'il existe une tournée de longueur au plus L alors, si la réponse est positive, il suffit d'exhiber une tournée qui vérifie cette propriété et il suffit alors de la parcourir, ce qui ne requiert qu'un temps proportionnel au nombre de villes.

L'ensemble des problèmes de décision que l'on peut résoudre en temps polynomial est noté P et il est généralement conjecturé que $P \neq NP$ (Delahaye, 2005 ; Sipser, 1992), autrement dit que certains problèmes de NP ne peuvent pas être résolus en temps polynomial. Cette conjecture fait partie des sept célèbres problèmes du millénaire dotés par la fondation Clay d'un prix d'un million de dollars [1] (Delvin, 2002).

Il y a bien sûr des problèmes de décision qui ne sont pas dans NP, certains problèmes de décision ne sont même pas décidables auquel cas il n'existe tout simplement pas d'algorithme permettant systématiquement d'obtenir une réponse en temps fini ! Le problème de l'arrêt d'une machine de Turing est emblématique de cette dernière famille (Turing, 1936 ; Turing & Girard, 1995).

Il existe dans NP des problèmes qui ont la particularité d'être polynomialement équivalents à tous les autres problèmes de NP, il s'agit des problèmes NP-complets. L'existence de tels problèmes est riche de conséquences : les problèmes NP-complets sont les plus difficiles de NP et s'il existe un algorithme polynomial qui permet de résoudre l'un des problèmes NP-complets alors cet algorithme peut être utilisé pour résoudre tous les problèmes de NP (NP-complets ou non) en temps polynomial.

Jusqu'ici nous avons uniquement considéré des problèmes de décision or nous sommes intéressés par des problèmes d'optimisation. Il est clair que si l'on sait résoudre un problème d'optimisation, par exemple, trouver une tournée de voyageur de commerce de longueur minimale alors on sait répondre à toutes les questions de la forme « existe-t-il une tournée de voyageur de commerce de longueur au plus L ? ». Inversement, si l'on sait répondre aux questions de la forme « existe-t-il une tournée de voyageur de commerce de longueur au plus L ? » alors on sait aussi résoudre le problème d'optimisation associé, par exemple, à l'aide d'une recherche dichotomique (Schrijver, 2004). Le nombre de questions à poser est alors de l'ordre du nombre de bits nécessaire pour coder les distances. Ceci permet donc de définir la notion de problème d'optimisation NP-difficile : un problème

1. Voir sur la toile à l'adresse `www.claymath.org/millennium/`.

d'optimisation NP-difficile est un problème d'optimisation qui est aussi difficile qu'un problème NP-complet.

1.1.4 Exemple : les problèmes d'ordonnancement

Afin d'illustrer notre propos, cette section présente succinctement quelques problèmes classiques d'ordonnancement.

Commençons par rappeler une définition, due à Carlier & Chrétienne (1988) :

> « Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. »

Typiquement, les données d'un problème d'ordonnancement sont des tâches et leurs caractéristiques (durée, date de disponibilité, date échue, caractère morcelable, etc.), des contraintes (par exemple des relations d'antériorité entre les tâches), des ressources consommables (argent, matières premières, etc.) ou renouvelables (machines, processeurs, fichiers, personnel, etc.) et une fonction économique (date de fin d'exécution, plus grand retard, retard moyen pondéré, etc.).

Dans la mesure où les problèmes d'ordonnancement se rencontrent dans de nombreux domaines industriels (informatique, télécommunications, gestion de production, etc.), l'ordonnancement est l'une des branches les plus appliquées de l'optimisation combinatoire [2] ; elle est aussi à l'origine de ses problèmes parmi les plus redoutables (Bjorndal et al., 1995).

Pour se fixer les idées, considérons l'exemple d'un problème d'atelier d'apparence élémentaire. L'atelier contient m machines distinctes et l'on suppose donné un ensemble de n travaux, chaque travail étant un ensemble de m tâches de durée donnée qui doivent être exécutées sur une machine différente dans un ordre prescrit. Il n'y a pas de contrainte de précédence sur des tâches de travaux différents. L'objectif est alors de trouver un ordonnancement des tâches qui respecte les contraintes de précédence et qui minimise la durée totale, définie comme la date d'achèvement de la tâche la plus tardive. Ce problème, NP-difficile, connu sous le nom de *job-shop scheduling problem* dans la littérature anglo-saxonne, est d'une extrême difficulté : à titre d'exemple (Carlier & Pinson, 1989) une instance ayant à peine 10 machines et 10 travaux, aujourd'hui un passage obligé pour toute nouvelle méthode (Brucker et al., 1994), est restée ouverte pendant plus de vingt ans !

Pour plus de détails sur l'ordonnancement, sur l'extraordinaire diversité de ses problèmes et sur les algorithmes permettant de s'y attaquer, nous nous permettons de renvoyer le lecteur aux classiques du domaine que sont les ouvrages de Carlier & Chrétienne (1988), de Brucker (2004) et de Brucker & Knust (2006).

2. Les premiers problèmes traités par B. Roy à la Séma n'étaient-ils justement pas des problèmes d'ordonnancement ? (Lesourne, 2000 ; Colasse & Pavé, 1997)

Nous recommandons aussi vivement les publications du Gotha (1993, 2004) [3], ainsi que quelques mémoires de thèses récentes telles celles de Sourd (2000), de Jouglet (2002) et de Tercinet (2004).

1.2 Résolution des problèmes difficiles

Cette section introduit brièvement les trois grandes stratégies de résolution des problèmes d'optimisation combinatoire difficiles, à savoir l'étude de cas particuliers pouvant être résolus efficacement, la résolution approchée à l'aide d'algorithmes polynomiaux et la résolution exacte à l'aide d'algorithmes dont la complexité est exponentielle.

1.2.1 Cas particuliers polynomiaux

Le caractère NP-difficile d'un problème concerne sa complexité *au pire*. Il est généralement possible de trouver des classes d'instances qui, elles, peuvent être traitées en temps polynomial.

C'est par exemple le cas pour le problème d'atelier que nous avons évoqué à la section 1.1.4 : le cas à 2 machines et n travaux comme le cas à m machines et 2 travaux peuvent être résolus efficacement (Gotha, 1993).

1.2.2 Algorithmes de résolution approchée

Une *méthode approchée* ou *heuristique* pour un problème combinatoire difficile est un algorithme, généralement efficace, qui fournit des solutions réalisables, tenant compte de la fonction économique, mais sans garantie d'optimalité.

De manière usuelle, on regroupe les méthodes approchées en deux grandes familles [4] :

- les *méthodes par construction progressive*, desquelles découlent des algorithmes qui ne construisent qu'une solution réalisable et qui à chaque étape de la construction font un ou plusieurs choix définitifs. On parle de *méthode gloutonne* lorsque le choix effectué est celui qui améliore le plus la fonction économique ;

3. Groupe de Recherche en Ordonnancement Théorique et Appliqué, voir sur la toile à l'adresse `www-poleia.lip6.fr/sourd/gotha/`.

4. Dans les deux cas, on distingue les méthodes déterministes des méthodes stochastiques, dans lesquelles intervient un certain degré de hasard. Alors même qu'il est ce que les ordinateurs ne peuvent produire (Li & Vitányi, 1997), le hasard joue un rôle fondamental en informatique tant théorique qu'appliqué. Citons Karp (1991) : ≪ *Often, the execution time or space of a randomized algorithm is smaller than that of the best deterministic algorithm that we know of for the same problem. [...] Often, the introduction of randomization suffices to convert a simple and naive deterministic algorithm with bad worst-case behavior into a randomized algorithm that performs well with high probability on every possible input.* ≫.

– les *méthodes par améliorations successives*, desquelles découlent des algorithmes qui construisent une séquence de solutions admissibles, deux solutions successives se trouvant généralement dans un voisinage l'une de l'autre.

Généralement, les méthodes par construction progressive sont fortement liées à la structure du problème. En conséquence, pour un problème donné, il est parfois possible de majorer une certaine distance à l'optimum par une borne indépendante de l'instance traitée [5].

Depuis le début des années quatre-vingt sont apparus des paradigmes de conception de méthodes par améliorations successives : les *métaheuristiques* (recuit simulé, recherche avec tabous, algorithmes génétiques et bien d'autres, voir Dréo et al., 2003 ainsi que le chapitre 6 de ce mémoire). Il s'agit généralement de variantes plus ou moins sophistiquées de l'algorithme de recherche locale. D'un intérêt indéniable pour le praticien, car il en découle généralement des algorithmes à la fois simples et performants, il convient néanmoins de souligner que ces méthodes sont selon les cas plus ou moins bien comprises sur le plan théorique : malgré des succès expérimentaux parfois remarquables, il est rarement possible de borner une quelconque distance à l'optimum que ce soit en moyenne ou dans le pire des cas.

1.2.3 Algorithmes de résolution exacte

Une *méthode exacte* pour un problème combinatoire difficile est un algorithme dont la complexité est exponentielle, mais qui fournit des solutions optimales.

Les méthodes exactes sont généralement arborescentes, on parle d'*algorithmes d'énumération par séparation et évaluation* (*branch-and-bound*, en anglais), leur succès reposant principalement sur la notion d'énumération implicite : des pans entiers de l'arborescence sont élagués par exemple lorsqu'une évaluation par défaut d'une solution partielle indique qu'il est impossible de la compléter de manière à fournir une solution de valeur moindre que la meilleure solution déjà rencontrée.

La figure 1.1 illustre ce principe. L'exemple correspond à l'exploration en profondeur d'abord de l'ensemble des permutations des éléments de l'ensemble $\{a, b, c, d\}$. La valeur indiquée à côté de chacun des nœuds correspond à l'évalua-

5. À titre d'exemple, pour le problème de conditionnement qui consiste, étant donnée n objets de poids $w_1, \ldots, w_n$ et m sacs de capacité C, à trouver une fonction $f : \{1, \ldots, n\} \longrightarrow \{1, \ldots, m\}$ qui minimise $\sum_{i=1}^{m} \max\left(C, \sum_{j=1:f(j)=i}^{n} w_j\right)$, on peut montrer (Dell'Olmo et al., 1998) que l'algorithme qui consiste simplement à trier les objets par ordre de poids décroissant et à mettre chaque objet dans le sac le moins rempli est tel que pour toute instance I du problème on a $\frac{\mathrm{LPT}(I)}{\mathrm{OPT}(I)} \leq \frac{13}{12} = 1.0833$, où $\mathrm{LPT}(I)$ et $\mathrm{OPT}(I)$ dénotent respectivement la valeur de la solution fournie par l'algorithme et celle d'une solution optimale. Ces résultats ont été utilisés par l'auteur pour résoudre un problème d'affectation de cellules radios à des liens de communication (Sirdey, 2004a, 2006a).

tion par défaut ou à la valeur de la solution, s'il s'agit d'une feuille. Les parties de l'arborescence en pointillés ne sont pas explorées explicitement. La première solution réalisable rencontrée, *abcd*, a pour valeur 10 et la solution *abdc*, de valeur 5 est obtenue dans la foulée. Dans la mesure où l'évaluation par défaut des permutations commençant par *ac* est égale à $6 > 5$ il n'est pas nécessaire d'aller plus loin dans l'exploration de cette branche. La solution optimale, *bcad*, de valeur 3, est obtenue plus tard. La branche des solutions commençant par *c* est intégralement élaguée, l'évaluation par défaut du nœud *c* étant égale à $4 > 3$.

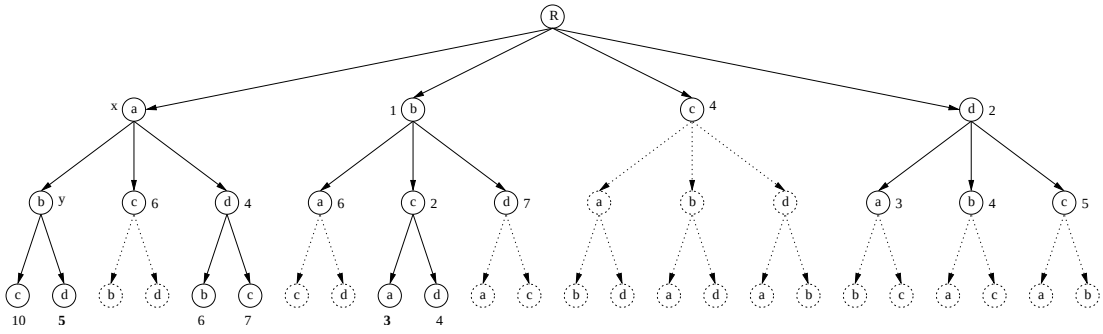


FIGURE 1.1 – Illusration du principe d'énumération implicite.

Les performances pratiques de ces algorithmes dépendent crucialement de la qualité de l'évaluation par défaut mais aussi de la présence de *règles de dominance*, règles qui permettent d'identifier et donc d'élaguer des branches redondantes de l'arborescence. Plus précisément, une règle de dominance est une contrainte qui peut être ajouté au problème initial sans changer la valeur de l'optimum, autrement dit, une contrainte telle qu'il existe au moins une solution optimale qui la vérifie.

Dans ce manuscrit, nous distinguons deux grandes familles d'algorithmes de recherche arborescente :

- les *algorithmes de recherche arborescente combinatoires* (aussi connus sous le nom d'*algorithmes par séparation et évaluation*), qui exploitent la structure du problème de manière directe ;
- les *algorithmes de recherche arborescente polyédriques* (aussi connus sous le nom d'*algorithmes de coupes et branchements*), qui sont basés sur la programmation linéaire et qui sont le sujet de la section 1.3.

1.2.4 Complémentarité des approches exacte et approchée

En présence d'un verrou théorique présumé, $P \neq NP$, des compromis s'imposent. De ce point de vue, les approches exactes et approchées ne sont que les deux faces d'une même pièce : en résolution exacte, le compromis favorise l'optimalité au détriment de garanties sur le temps de calcul et, en résolution

approchée, c'est exactement l'inverse, à savoir que le temps de calcul est mis en avant au détriment de garanties d'optimalité.

D'un côté, les algorithmes de résolution approchée servent à attaquer des instances de grande taille mais aussi, généralement, à fournir une bonne solution initiale pour une méthode exacte, ce qui permet déjà un premier élagage significatif de l'arborescence de recherche. D'un autre côté, les méthodes exactes permettent à la fois de résoudre certaines instances et de valider empiriquement la capacité des algorithmes approchés à justement fournir de bonnes solutions. Bien sûr, une méthode exacte avec un critère d'arrêt prématuré est une méthode approchée.

C'est dire si les deux approches sont complémentaires !

Aussi, dans les deux cas, l'expérimentation joue comme en sciences physiques un rôle prépondérant : les constructions théoriques les plus élégantes, dont l'exemple des coupes de Gomory (1958) est l'archétype, comme les heuristiques les plus *ad hoc*[6] doivent sans exception y être confrontées, souvent à leur détriment.

1.3 L'approche polyédrale

Cette section vise à présenter les grands principes de l'approche polyédrale, omniprésente dans nos travaux, ainsi que les quelques notions fondamentales de la théorie des polyèdres qui nous seront utiles tout au long de ce mémoire.

Pour plus de détails, nous nous permettons de renvoyer le lecteur aux classiques du domaine que sont les ouvrages (par ordre d'accessibilité croissante) de Schrijver (1986), de Nemhauser & Wolsey (1999) ainsi que de Maurras (2002), notamment.

Nous recommandons aussi sans réserve les deux articles introductifs d'Aardal & van Hoesel (1996, 1999) ainsi que celui de Mahjoub (2005)[7].

1.3.1 Principe

Un grand nombre de problèmes d'optimisation combinatoire, NP-difficiles ou non, peuvent être mis sous la forme d'un programme linéaire en nombres entiers, c'est-à-dire d'un programme mathématique de la forme

$$\begin{cases} \text{Minimiser } c^T x, \\ \text{s. l. c.} \\ Ax \leq b, \\ x \in \mathbb{Z}^n. \end{cases} \tag{1.1}$$

6. Au sens péjoratif du terme (Glass, 2002).

7. Pour une introduction en douceur, le lecteur non spécialiste peut aussi se reporter à l'article de vulgarisation, à paraître dans le journal *L'Ouvert*, que nous avons écrit (Sirdey, 2007c).

En termes géométriques, il s'agit, tout comme en programmation linéaire continue, d'optimiser une forme linéaire sur un polyèdre, l'optimum (s'il existe) étant forcément réalisé en l'un de ses sommets. Malheureusement, pour les problèmes NP-difficiles, la structure de ce polyèdre est d'une complexité à toute épreuve. Ceci étant dit, une connaissance même très partielle de cette structure s'avère, en pratique, extrêmement pertinente.

En résolution exacte des problèmes NP-difficiles, l'approche polyédrale consiste à exploiter, dans le cadre d'un schéma de recherche arborescente, la *relaxation linéaire*, c'est-à-dire le programme linéaire continu obtenu en ignorant la contrainte d'intégrité (1.1), itérativement enrichie de contraintes spécialisées qui permettent de « couper » les solutions fractionnaires successives. Ces contraintes définissent généralement des facettes, faces propres de dimension maximale, du polyèdre, appelé *polyèdre entier*, défini comme l'enveloppe convexe des solutions entières du système $Ax \leq b$.

La figure 1.2 illustre le principe de coupe : P est le polyèdre $\{x \in \mathbb{R}^2 : Ax \leq b\}$ et P_I est le polyèdre entier correspondant, la facette F sépare le sommet fractionaire $x \in P$ de P_I.

1.3.2 Indépendances linéaire et affine, enveloppe convexe

Soit $x^{(1)}, \ldots, x^{(m)}$ un ensemble de m points de $\mathbb{R}^n$. Un point $y \in \mathbb{R}^n$ est une *combinaison linéaire* de ces points s'il existe n réels $\lambda_1, \ldots, \lambda_n$ tels que

$$y = \sum_{i=1}^{m} \lambda_i x^{(i)}.$$

Si, de plus, $\sum_{i=1}^{m} \lambda_i = 1$ alors y est une *combinaison affine* des points $x^{(1)}, \ldots, x^{(m)}$. Enfin, on parle de *combinaison conique* lorsque $\lambda_i \geq 0$ pour tout $i \in \{1, \ldots, m\}$ et de *combinaison convexe* lorsqu'en plus $\sum_{i=1}^{m} \lambda_i = 1$. Ces notions sont illustrées sur la figure 1.3.

Des points de $\mathbb{R}^n$ sont dits *linéairement indépendants* (respectivement *affinement indépendants*) s'ils ne sont pas des combinaisons linéaires (respectivement affines) les uns des autres.

Soit S un ensemble non vide de points de $\mathbb{R}^n$. L'*enveloppe convexe* de S, noté $\text{conv}(S)$ correspond à l'ensemble des points de $\mathbb{R}^n$ qui sont des combinaisons convexes des points de S.

1.3.3 Polyèdre, polytope et dimension

Un *polyèdre* P de $\mathbb{R}^n$ est défini comme l'intersection d'un ensemble fini de demi-espaces fermés,

$$P = \bigcap_{i=1}^{N} D_i,$$

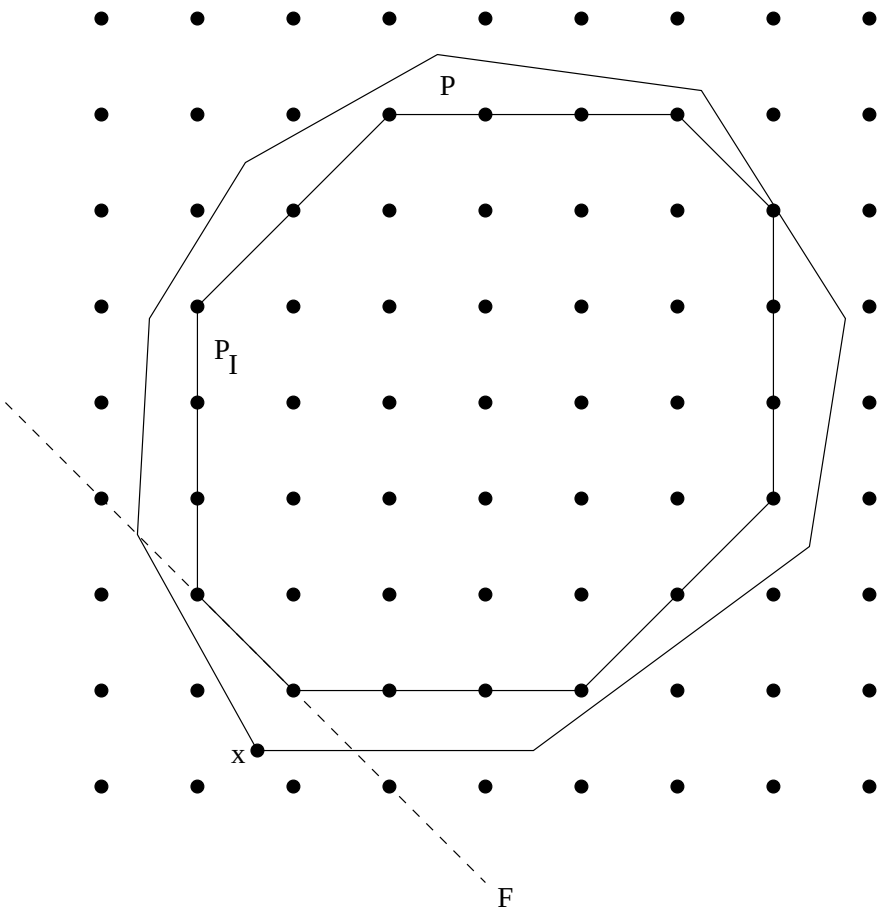
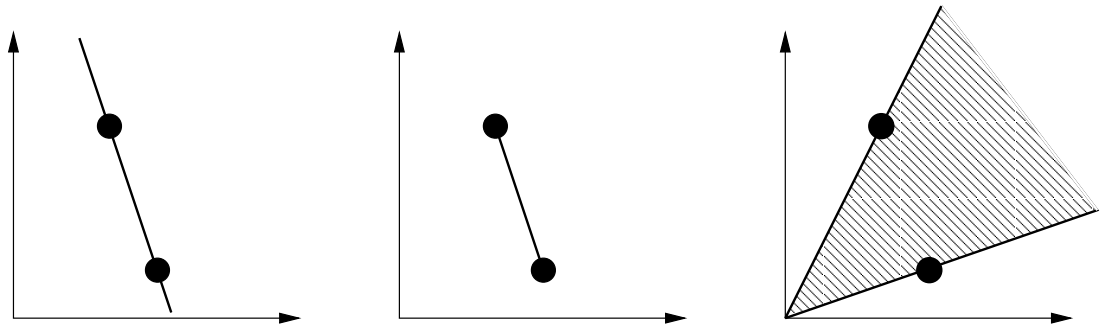


FIGURE 1.2 – Illusration du principe de coupe.



(a) Enveloppe affine. (b) Enveloppe convexe. (c) Enveloppe conique.

FIGURE 1.3 – Enveloppes affine, convexe et conique d'un ensemble de deux points.

ou, de manière équivalente,

$$P = \{x \in \mathbb{R}^n : Ax \leq b\},$$

avec $D_i = A_{i,\cdot}x \leq b_i$.

Soit $k+1$ le nombre maximum de points affinement indépendants de P, alors la *dimension* de P, noté $\dim(P)$, est égale à k. Lorsque $\dim(P) = n$, on dit que P est de *pleine dimension*.

Un polyèdre borné (*i. e.* contenu dans une boule de rayon fini) s'appelle un *polytope*. Tout polytope correspond à l'enveloppe convexe de ses sommets.

1.3.4 Inégalités valides, faces et facettes

Soit P un polyèdre contenu dans $\mathbb{R}^n$, une inégalité $a^T x \leq \alpha$ ($a \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$) est dite *valide* pour P si

$$P \subseteq \{x \in \mathbb{R}^n : a^T x \leq \alpha\}.$$

Si, de plus,

$$P \cap \{x \in \mathbb{R}^n : a^T x = \alpha\} \neq \emptyset$$

alors on dit que l'hyperplan $H = \{x \in \mathbb{R}^n : a^T x = \alpha\}$ *supporte* P. Le polyèdre $P \cap H$ définit alors une *face* de P. Un polyèdre P comporte deux autres faces : $\emptyset$ et P lui même.

Les faces de dimension 0 et 1 d'un polyèdre P s'appellent respectivement des *sommets* et des *arêtes*, les faces de dimension maximale (à l'exception de P lui même) s'appellent des *facettes*. Une facette contient $\dim P$ points affinement indépendants.

Étant donnée une inégalité valide, $a^T x \leq \alpha$ ($a \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$), pour un polytope P de $\mathbb{R}^n$ comment prouver qu'il s'agit bien d'une facette ? La technique de preuve que nous avons le plus souvent utilisée consiste essentiellement à appliquer la définition : montrer qu'il existe $\dim P$ points affinement indépendants de P qui appartiennent à l'hyperplan $a^T x = \alpha$. D'autres techniques sont répertoriées dans Mahjoub (2005).

À titre d'exemple, une technique fort commode pour montrer qu'un point $y \in \mathbb{R}^n$ est affinement indépendant d'un ensemble $x^{(1)}, \ldots, x^{(m)}$ de m points de $\mathbb{R}^n$ consiste à identifier un vecteur $b \in \mathbb{R}^n$ tel que $b^T x^{(i)} = \beta$, pour $i = 1, \ldots, m$, et que $b^T y = \gamma$ avec $\beta \neq \gamma$. Supposer que y peut s'écrire comme combinaison affine des $x^{(i)}$ conduit alors tout simplement à la contradiction :

$$\gamma = \sum_{j=1}^{n} b_j y_j = \sum_{j=1}^{n} b_j \sum_{i=1}^{m} \lambda_i x_j^{(i)} = \sum_{i=1}^{m} \lambda_i \underbrace{\sum_{j=1}^{n} b_j x_j^{(i)}}_{=\beta} = \beta \underbrace{\sum_{i=1}^{m} \lambda_i}_{=1} = \beta.$$

1.3.5 Polyèdres et voyageur de commerce

La pertinence pratique des algorithmes issus de l'approche polyédrale n'est plus à démontrer.

En particulier, c'est à l'aide d'un algorithme de recherche arborescente polyédrique, bien entendu particulièrement raffiné, et d'une bonne dose d'ingéniosité informatique qu'Applegate et al. (1998) ont réussi à résoudre une instance du problème du voyageur de commerce à plus de 13 000 villes. Certes au prix de l'équivalent d'une dizaine de jours de calcul réparti sur un réseau d'une cinquantaine de stations de travail, ce qui ne diminue en rien la performance. La même équipe détient le présent record : il est, depuis mai 2004, possible d'effectuer une tournée optimale des 24 978 villes de Suède [8] !

Soulignons néanmoins qu'en dépit de ces résultats empiriques remarquables, on sait aussi construire des instances de petite taille que l'on n'arrive pas, en pratique, à résoudre (Carlier, 2006). Le problème du voyageur de commerce reste donc difficile, dans le pire des cas, et la conjecture $P \neq NP$ tient bon.

8. Voir sur la toile à l'adresse `www.tsp.gatech.edu`.

Chapitre 2

Contexte industriel

Introduction

Ce chapitre présente le contexte industriel de ce travail de recherche. Nous commençons par donner un aperçu de l'architecture globale d'un réseau GSM ainsi que des fonctions des deux principaux composants du sous-système radio : les stations de base et les contrôleurs de stations de base. Nous esquissons ensuite l'architecture matérielle du BSCe3, autocommutateur réparti auquel nos travaux s'appliquent directement, et présentons les principes de gestion des stations de base. Ceci nous conduit à définir la notion d'application de traitement d'appels. Le chapitre se conclut par une présentation succincte des mécanismes de tolérance aux pannes du BSCe3 et par un premier contact avec le problème de reconfiguration qui sera formalisé au chapitre 3.

Il est bien connu que le monde des télécommunications et, en particulier, de la téléphonie mobile fait une utilisation importante d'acronymes. Chaque acronyme est défini au moment de sa première utilisation.

2.1 Rappels sur les réseaux GSM

Cette section rappelle les connaissances de base sur le système GSM (*Global System for Mobile communications*) nécessaires à une bonne compréhension du contexte industriel de ce travail. Par souci de concision, nous avons décidé d'occulter les équipements liés au SMS (*Short Message Service*) et au GPRS (*General Packet Radio System*), sur ce sujet, et pour plus de détails, nous nous permettons d'orienter le lecteur vers les classiques de la littérature spécialisée que sont par exemple les ouvrages de Lagrange et al. (2000) et de Mouly & Pautet (1992), ouvrages desquels la présente section est largement inspirée.

2.1.1 Architecture des réseaux GSM

Fonctionnellement, un réseau GSM est construit autour des entités logiques suivantes : le sous-système radio ou BSS (*Base Station Subsystem*), le sous-système d'acheminement ou NSS (*Network and Switching Subsystem*) et le sous-système d'exploitation et de maintenance ou OSS (*Operation SubSystem*). Cette section indique les principales caractéristiques de chacune de ces entités. Voir aussi la figure 2.1.

Le sous-système radio assure les transmissions radioélectriques et gère la ressource radio. Le BSS est principalement constitué de stations de base ou BTS (*Base Transceiver Sation*), déployées dans toute la zone à desservir, ainsi que de contrôleurs de stations radios ou BSC (*Base Station Controller*). Une station de base est en général associée à une ou plusieurs cellules (jusqu'à six). Un BSC gère plusieurs BTS et permet une première concentration du trafic. Dans la mesure où, sur l'interface radio, la voix est codée sur 13 kbits/s et que le sous-système d'acheminement gère des circuits à 64 kbits/s, il convient de réaliser un transcodage 13 kbits/s vers 64 kbits/s des canaux de voix. Cette tâche incombe aux transcodeurs ou TRAU (*Transcoder/Rate Adapter Unit*) que l'on appelle TCU (*TransCoding Units*) dans l'implémentation Nortel.

Le sous-système d'acheminement comprend l'ensemble des fonctions nécessaires à l'établissement des appels et à la mobilité. Le NSS est principalement constitué de bases de données et de commutateurs. Les commutateurs mobiles ou MSC (*Mobile Switching Center*) associés aux VLR (*Visitor Location Register*) effectuent la gestion des appels, le HLR [1] (*Home Location Register*) se charge du stockage des données de localisation et de caractérisation des abonnés. À ces équipements s'ajoutent une base de données appelée EIR (*Equipment Identity Register*) qui contient les identités des terminaux ainsi que le centre d'authentification ou AUC[1] (*AUthentication Center*) qui mémorise, pour chaque abonné, les données nécessaires aux procédures d'authentification et d'établissement des clefs de chiffrement.

Le sous-système d'exploitation et de maintenance permet à l'opérateur d'exploiter son réseau, il est composé d'un centre de gestion ou NMC (*Network Management Center*) qui pilote plusieurs types de centres d'exploitation et de maintenance ou OMC (*Operation and Maintenance Center*) tels que les OMC-R (pour les équipements du sous-système radio) ou les OMC-S (pour les équipements du sous-système d'acheminement).

2.1.2 Les stations de base

Succinctement, une station de base peut être vue comme un ensemble d'émetteurs-récepteurs ou TRX. Une BTS a notamment la charge de la couche physique

1. C'est d'ailleurs sur le HLR et l'AUC que l'auteur a commencé sa carrière, fin 1998, dans l'ouest londonien, avant de rejoindre l'équipe BSC en juillet 2001.

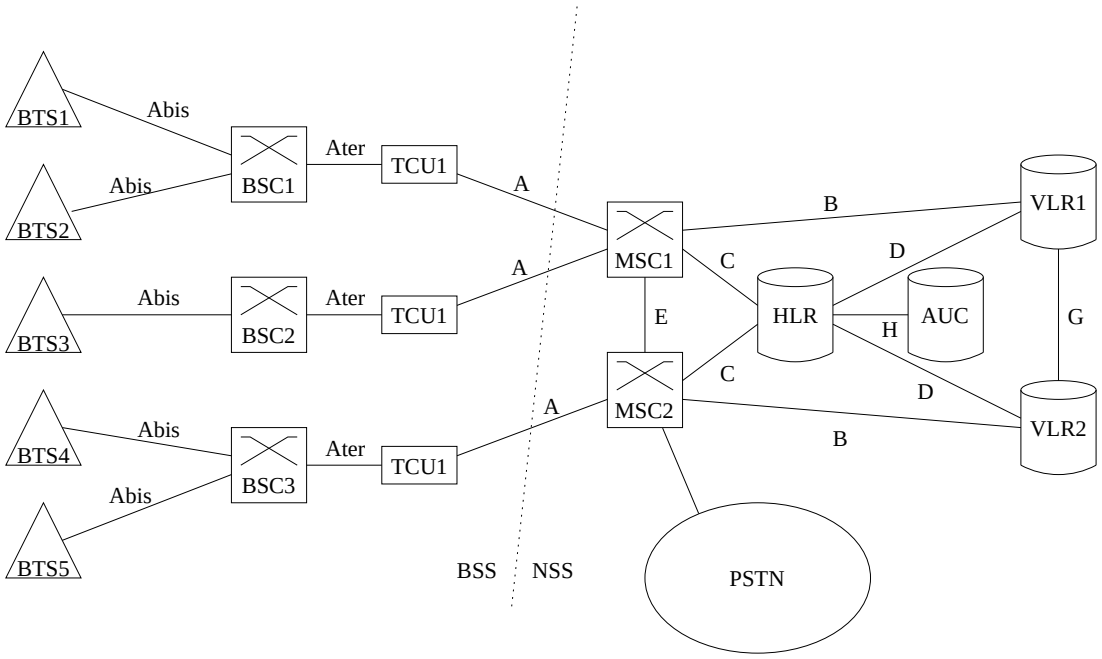


FIGURE 2.1 – Représentation simplifiée de l'architecture d'un réseau GSM (le sous-système d'exploitation et de maintenance a été omis). Le PSTN (*Public Switched Telephone Network*) correspond au réseau téléphonique commuté public.

sur l'interface Air, c'est-à-dire la transmission radio (modulation, démodulation, égalisation et codage correcteur d'erreur), le multiplexage TDMA (*Time Division Multiple Access*), les sauts de fréquence lents et le chiffrement. Aussi, une station de base gère la couche liaison de données par le biais de laquelle est acheminée la signalisation sur l'interface Air (protocole LAPDm) et sur l'interface Abis (protocole LAPD). Enfin, une BTS réalise l'ensemble des mesures nécessaires à l'évaluation de la qualité d'une communication en cours, ces mesures sont exploitées par les BSC en relation avec les transferts de communication intercellulaires (*handover*).

2.1.3 Le contrôleur de stations de base

Le contrôleur de stations de base a pour fonction principale de gérer la ressource radio c'est-à-dire qu'il commande l'allocation des canaux, qu'il utilise les mesures fournies par les BTS pour contrôler la puissance d'émission du mobile et/ou de la station de base et qu'il prend les décisions de transfert de communication intercellulaire. C'est aussi un commutateur qui réalise une première concentration des circuits vers le MSC. Le BSC est relié aux BTS et au MSC par des liens MIC, les couches liaison de données avec les BTS et le MSC se basent respectivement sur les protocoles LAPD et SS7.

2.2 Le BSCe3

Le BSCe3 (Architecture BSC, 2003a) est la nouvelle génération de BSC commercialisée par Nortel, il répond aux demandes du marché en termes de capacité, de connectivité et de souplesse d'utilisation. En particulier (Architecture BSC, 2003b), il peut piloter jusqu'à 500 BTS et supporter un trafic offert allant de 600 à 3000 erlangs (pour un taux de blocage d'appel de 0.1%, voir la section 2.2.3), la montée en charge se réalise par ajout d'unités de traitement d'appels.

En raison des contraintes imposées à la disponibilité du système (Sirdey, 2004b ; Hudepohl et al., 2004) ainsi qu'à son caractère réparti, le BSCe3 doit être capable de résister aux défaillances de certains de ses composants. Ceci nécessite la mise en place de mécanismes matériels et logiciels dédiés à la tolérance aux pannes.

Le BSCe3 est un équipement monocabinet composé de deux sous-systèmes : le sous-système de contrôle et le sous-système d'interface que nous allons brièvement décrire.

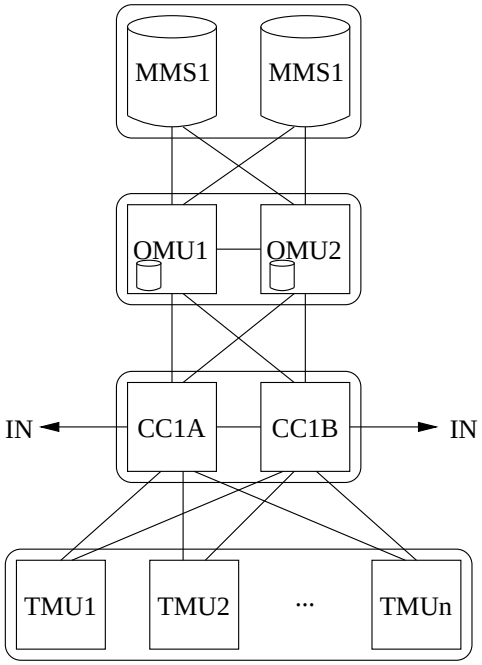


FIGURE 2.2 – Aperçu de l'architecture matérielle du sous-système de contrôle du BSCe3.

2.2.1 Le sous-système de contrôle

Le sous-système de contrôle ou CN (*Control Node*) est composé des sous-systèmes suivants (voir la figure 2.2).

Le sous-système d'opération et de maintenance qui comprend deux unités d'opération et de maintenance ou OMU (*Operation and Maintenance Unit*). Ce sous-système est entre autres responsable de la gestion des équipements du sous-système de contrôle, du contrôle du sous-système d'interface, de la gestion des disques et de la gestion du lien avec l'OMC-R. En termes de fiabilité les deux OMU fonctionnent en redondance passive logicielle (Guerraoui & Schiper, 1997 ; Jalote, 1994). Dans le cas d'un système en fonctionnement normal l'une des deux OMU est active et gère les processus responsables du bon fonctionnement du système, ces derniers sont répliqués sur l'autre OMU où leurs équivalents passifs se contentent de recevoir les messages de mise à jour (en provenance des processus actifs) par le biais desquels la synchronisation nécessaire à une reprise d'activité fiable en cas de défaillance de l'OMU active est préservée.

Le sous-système de stockage est constitué de deux unités de stockage ou MMS (*Memory Mass Storage*). Ce sous-système est responsable de la mémorisation des données qui doivent survivre à une défaillance de l'OMU active. En termes de fiabilité le sous-système de stockage est un système parallèle : ce n'est qu'à partir du moment où les deux unités sont défaillantes que le système est en panne.

Le sous-système de commutation ATM [2] comprend deux commutateurs ATM ou CC1, son rôle consiste principalement à permettre aux différents composants du sous-système de contrôle de communiquer entre eux et avec le sous-système d'interface. En termes de fiabilité le sous-système de commutation ATM est un système parallèle : dans le cas d'un système en fonctionnement normal l'intégralité du trafic ATM est dupliqué sur les deux CC1 et c'est aux unités connectées à ce sous-système qu'il incombe d'extraire un flux de cellules cohérent à partir des flux provenant de chacun des deux CC1. La défaillance de l'un des deux commutateurs n'induit aucune modification du comportement ou de la charge de l'autre.

Le sous-système de traitement d'appels contient au moins 4 et au plus 14 unités de traitement d'appels ou TMU (*Traffic Management Units*), comme son nom l'indique, le rôle de ce sous-système est de gérer les applications de traitement d'appels. En termes de fiabilité le sous-système de traitement d'appels peut grossièrement être considéré comme un système à redondance active k sur n. Dans la mesure où ce travail s'intéresse principalement au sous-système de contrôle et à la gestion de sa charge de traitement d'appels nous reviendrons longuement sur les mécanismes de protection de cette dernière (en particulier à la section 2.3).

La figure 2.3 présente un diagramme de fiabilité simplifié du sous-système de contrôle.

2. Le lecteur intéressé par plus de détails sur la technologie ATM peut se reporter à Tanenbaum (1996).

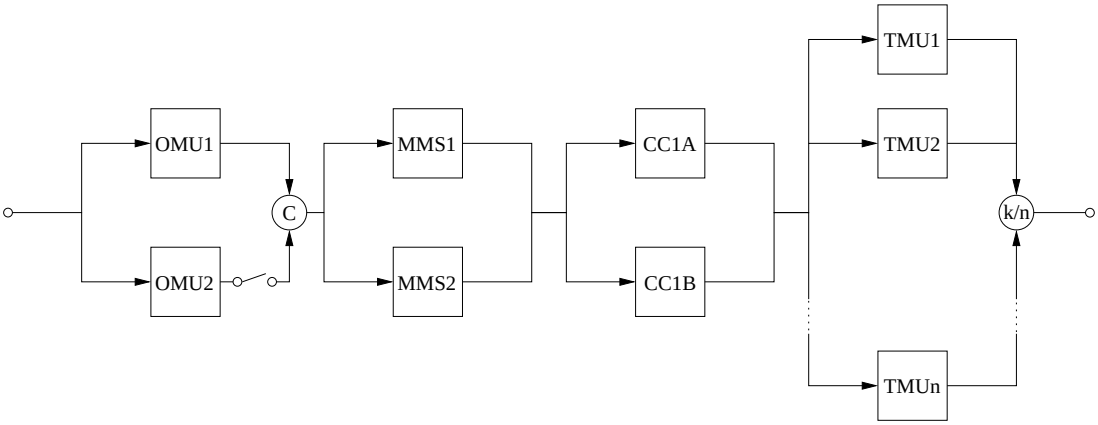


FIGURE 2.3 – Diagramme de fiabilité simplifié du sous-système de contrôle du BSCe3. Les conventions sont celles de l'ouvrage de Pagès & Gondran (1980).

2.2.2 Le sous-système d'interface

Le sous-système d'interface ou IN (*Interface Node*) est un brasseur dont le rôle se résume à la connexion des canaux MIC de l'interface Abis (en provenance et à destination des BTS) avec ceux de l'interface A-ter (en provenance et à destination des TCU). L'architecture matérielle du sous-système d'interface est beaucoup plus simple que celle du sous-système de contrôle et il en est de même concernant les mécanismes dédiés à la tolérance aux pannes. Dans la mesure où ce travail ne s'intéresse quasiment qu'au sous-système de contrôle nous ne fournissons pas plus de précisions.

2.2.3 Gestion des stations de base

Cette section présente les éléments de télétrafic nécessaires à la définition des principales ressources offertes par le système et consommées par les applications de traitement d'appels. Pour plus de détails, le lecteur est par exemple orienté vers les publications d'Hébuterne (1985) ou de Songhurst (1997).

Sous des hypothèses classiques et assez bien vérifiées en pratique (système fonctionnant à appels perdus soumis à un flux d'arrivées obéissant à la loi de Poisson et temps de prise exponentiels), il est possible de calculer le trafic offert ou *offered traffic* (c'est-à-dire le nombre moyen d'appels arrivant pendant le temps moyen de service), noté A, par chaque cellule. Ce calcul se réalise par inversion de la célèbre formule d'Erlang,

$$B = \frac{\frac{A^N}{N!}}{\sum_{i=0}^{N} \frac{A^i}{i!}},$$

où B et N dénotent respectivement le taux de blocage d'appel (c'est-à-dire la proportion d'appels rejetés) et le nombre de canaux voix de la cellule (qui se

déduit du nombre de TRX qui la constituent). Le trafic écoulé ou *carried traffic* (c'est-à-dire le nombre moyen de circuits occupés simultanément), noté A_e, se déduit de la formule $B = \frac{A-A_e}{A}$. Enfin, on calcule le trafic offert (respectivement écoulé) par une station de base en sommant les trafics offerts (respectivement écoulés) par chacune de ses cellules.

Cette notion de trafic offert est importante dans la mesure où elle est à la base du dimensionnement des unités de traitement d'appels et donc du système : une TMU peut offrir un nombre d'erlangs C si elle possède les ressources nécessaires (en termes de connectivité, de capacité de calcul et de mémoire) pour supporter confortablement la signalisation induite par un trafic offert de C erlangs selon un modèle de trafic « agressif »[3]. Lorsque le trafic réel sur l'ensemble des stations de base géré par une unité de traitement d'appels dépasse temporairement sa capacité, des mécanismes de contrôle de congestion sont activés afin que la surcharge ne se traduise pas par une défaillance.

De manière à limiter le nombre de processus gérés par les TMU, l'ensemble des stations de base sous la responsabilité du système est initialement partitionné en groupes de cellules[4] ou CG (*Cell Group*). Chacun de ces groupes définit une application de traitement d'appels. Naturellement, le trafic offert (respectivement écoulé) par un groupe de cellules est la somme des trafics offerts (respectivement écoulés) par les stations qui le composent.

Afin d'acheminer la signalisation nécessaire au déroulement des procédures GSM, il convient de réserver un nombre de canaux LAPD pour chacune des stations de base (ce nombre dépend à la fois du nombre de cellules et du nombre total de TRX qui la composent). Les groupes de cellules consomment donc les canaux LAPD offerts par les unités de traitement d'appels. Il convient par ailleurs de souligner que la limite de capacité en termes de canaux LAPD est une contrainte forte (contrairement à la limite de capacité en erlangs que l'on pourrait considérer légèrement dépassable de manière transitoire).

2.3 Tolérance aux pannes

Cette section fournit un aperçu des mécanismes de gestion de la charge du système en particulier en ce qui concerne sa protection contre les défaillances des unités de traitement d'appels et sa répartition sur ces dernières.

3. Typiquement, un modèle de trafic consiste en la donnée d'une série de paramètres, notamment, un taux d'arrivée (nombre moyen d'appels se présentant par unité de temps) pour l'heure pleine et des taux de transfert de communication intercellulaire.

4. L'appellation est légèrement abusive dans la mesure où il s'agit de groupe de stations de base.

2.3.1 Principes de la tolérance aux pannes

La tolérance aux pannes, définie comme la capacité d'un système à fonctionner, potentiellement de manière dégradée, malgré la défaillance d'un ou plusieurs de ses constituants, est un enjeu majeur en conception des systèmes répartis. En effet, dès lors que l'on considère un nombre de composants suffisamment grand, la probabilité qu'ils ne soient pas tous opérationnels à un instant donné devient non négligeable. Ceci sans même tenir compte des aspects liés au logiciel qui peuvent, selon l'architecture retenue, entraîner des défaillances en cascade.

L'approche en redondance passive logicielle (Guerraoui & Schiper, 1997) a été retenue afin d'assurer la protection de la charge de traitement d'appels du sous-système de contrôle du BSC. De ce fait, chaque application de traitement d'appels est représentée, si tant est que la capacité disponible le permette, par un processus actif et un processus passif, synchronisé au démarrage puis mis à jour au fil de l'eau par l'actif, qui tournent sur des TMU bien évidemment distinctes. Un processus passif peut prendre la main à tout instant, sans interruption des appels dits stables (c'est-à-dire à l'exclusion des appels en cours d'établissement ou en cours de transfert intercellulaire) gérés par l'actif[5].

Le logiciel de gestion de la tolérance aux pannes peut piloter, depuis l'OMU, les opérations suivantes : démarrage d'un processus actif ; démarrage d'un processus passif dûment synchronisé avec le processus actif associé (naturellement, cette opération nécessite que l'application concernée soit au préalable représentée par un processus actif) ; basculement de l'activité d'un processus passif (ceci requiert la disparition préalable du processus actif correspondant) ; migration (avec impact négligeable sur le service) d'un processus actif d'une TMU vers une autre ; migration d'un processus passif ; arrêt d'un processus actif ; arrêt d'un processus passif.

Aussi, il convient de souligner qu'une migration de processus actif requiert l'arrêt temporaire du processus passif correspondant. En effet, une telle migration consiste à démarrer un pseudo-passif (dûment synchronisé puis mis à jour au fil de l'eau) sur la TMU cible, puis à arrêter l'actif et, enfin, à donner la main au pseudo-passif qui devient alors actif. L'arrêt temporaire du passif permet d'éviter la mise en œuvre de protocoles de resynchronisation visant à résoudre les ruptures de séquence telles que celle illustrée sur la figure 2.4. En effet, sur cet exemple, le nouvel actif applique les opérations A et B dans l'ordre BA alors que le passif les applique dans l'ordre AB ce qui peut conduire à de graves incohérences dans certains cas.

5. Nous avons occulté, ici, certaines difficultés liées au maintien d'un certain degré de cohérence globale du système. En particulier, le maintien de certaines propriétés de vivacité dans un contexte où l'allocation de bout en bout d'une ressource (par exemple, un circuit) résulte de la collaboration de plusieurs entités asynchrones pouvant tomber en panne à tout instant requiert la mise en œuvre de mécanismes de resynchronisation. De tels mécanismes ont été spécifiés et formellement vérifiés dans le cadre de la thèse de F. Derepas (2002)

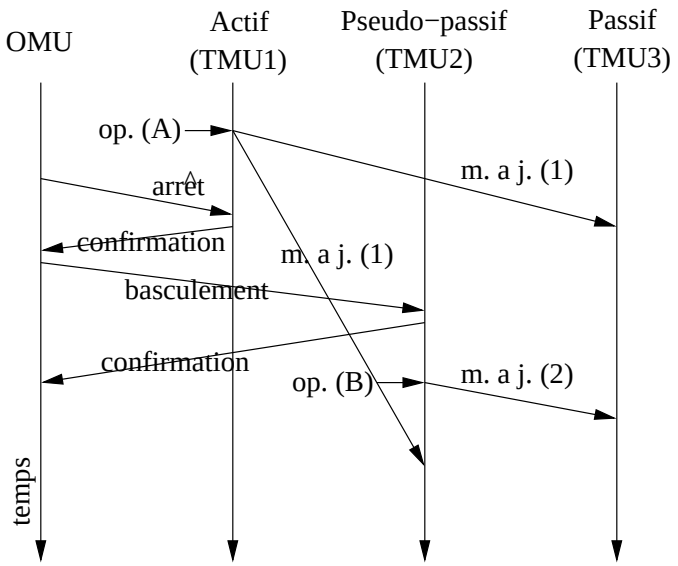


FIGURE 2.4 – Exemple de rupture de séquence justifiant l'arrêt temporaire du passif durant une migration d'actif.

2.3.2 États admissibles

Un état admissible du système consiste en une affectation des processus actifs et passifs telle que les contraintes de capacité sur les TMU ne soient pas violées et telle qu'il n'y ait pas de passif sans actif.

Dans la mesure où la charge de traitement d'appel n'est protégée que contre des défaillances TMU suffisamment espacées dans le temps (sinon quoi il faudrait plusieurs passifs par actif, et assumer les conséquences de ce choix sur le substrat de communication [6]), la réserve de capacité à prévoir pour s'assurer de la faisabilité du basculement des passifs ne dépend pas linéairement de la consommation des actifs correspondants. À chaque processus passif est associé une consommation dite *effective*, qui représente une fraction, potentiellement nulle, de la consommation de l'actif (10% pour les erlangs [7], 0 pour les LAPD), ainsi qu'une consommation *provisionnelle*, qui correspond au reste. La consommation de ressources sur une TMU correspond donc à la somme des consommations des actifs à laquelle s'ajoute la somme des consommations effectives des passifs et la plus grande somme des consommations provisionnelles de passifs d'actifs tournant sur la même autre TMU. Plus formellement, soient A l'ensemble des actifs et P l'ensemble des passifs, la consommation d'une ressource sur la TMU t est donnée

6. *Grosso modo*, il faut alors une primitive de diffusion atomique (à moins d'accepter de redémarrer/resynchroniser les passifs à des instants bien choisis, voir Sirdey, 2007b).

7. En effet, les processus passifs ne sont pas totalement inactifs : ils traitent les messages de mise à jour que leur envoient en permanence les processus actifs qu'ils assurent.

par

$$\sum_{a\in A:f^{(A)}(a)=t} w_a + \sum_{p\in P:f^{(P)}(p)=t} w_p + \max_{t'\neq t} \sum_{\substack{p\in P:f^{(P)}(p)=t \\ \wedge f^{(A)}(\alpha(p))=t'}} w'_p,$$

où w_a, w_p et w'_p dénotent respectivement la consommation de l'actif a et les consommations effective et provisionnelle du passif p dans la ressource, où $f^{(A)} : A \longrightarrow T$ et $f^{(P)} : P \longrightarrow T$ dénotent respectivement les affectations des actifs et des passifs à l'ensemble des TMU, noté T, et, enfin, où $\alpha : P \longrightarrow A$ est telle que $\alpha(p)$ indique l'actif du passif p. Remarquons que $w_{\alpha(p)} = w_p + w'_p$.

Pour une formalisation plus détaillée, nous nous permettons de renvoyer le lecteur à Sirdey et al. (2003).

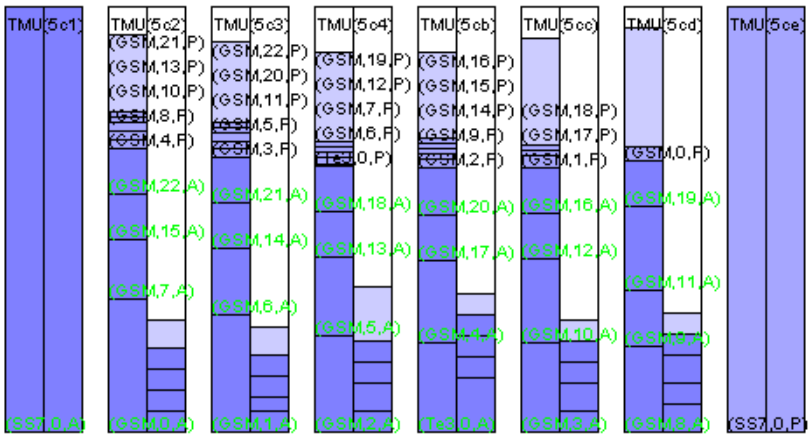


FIGURE 2.5 – Exemple d'état admissible, après démarrage du système.

La figure 2.5 illustre la notion d'état admissible. Une TMU y est représentée par un rectangle avec, en haut, son identifiant (par exemple 0x5c2). Pour une TMU, la colonne de gauche correspond à la ressource erlang et la colonne de droite aux ports LAPD. Les identifiants des applications de traitement d'appels sont rattachés au bas du rectangle représentant leur consommation en erlangs. Par exemple, l'actif de l'application GSM0 tourne sur la TMU 0x5c2 et le passif correspondant sur la TMU 0x5cd. Un rectangle bleu foncé représente la consommation effective d'un actif, un rectangle bleu représente la consommation effective d'un passif et un rectangle bleu clair (il n'y en a qu'un par TMU) représente la superposition des consommations provisionnelles des passifs qui tournent sur la TMU.

2.3.3 Taux de remplissage

Considérons le cas à une ressource. On appelle *rapport passif/actif* et l'on note γ, le rapport entre la consommation effective d'un processus passif et celle

du processus actif qu'il assure, par exemple, 0.1 pour la ressource erlang.

À partir du nombre d'unités qui composent le système, noté n, de la capacité d'une unité, notée C, et du rapport passif/actif, on peut déduire un *taux de remplissage* du système qui est égal à $\tau = \frac{x}{C}$ où x est la solution de l'équation

$$x + \gamma x + \frac{1-\gamma}{n-1}x = C, \tag{2.1}$$

soit

$$\tau = \frac{n-1}{n(\gamma+1)-2\gamma}. \tag{2.2}$$

La table 2.1 indique les taux de remplissage pour l'ensemble des configurations possibles du système.

n	$\gamma = 0.1$	$\gamma = 0.0$
2	0.5000	0.5000
3	0.6452	0.6667
4	0.7143	0.7500
5	0.7547	0.8000
6	0.7813	0.8333
7	0.8000	0.8571
8	0.8140	0.8750
9	0.8247	0.8889
10	0.8333	0.9000
11	0.8403	0.9091
12	0.8462	0.9167
13	0.8511	0.9231
14	0.8553	0.9286

TABLE 2.1 – Taux de remplissage.

Afin de justifier l'équation (2.1) considérons le cas où la distribution des processus est parfaitement équilibrée et où la consommation des processus actifs est égale à x sur chacune des unités. Pour un système en redondance $(n-1)$ sur n la consommation des processus passifs sur chacune des unités est égale à $(n-1)\frac{\gamma x}{n-1}$ (les autres unités assurent à parts égales la redondance d'une unité). Enfin, il convient de garantir qu'en cas de défaillance de l'une quelconque des unités la capacité résiduelle des autres unités est suffisante pour que les processus passifs qui assurent les processus actifs qui tournaient sur l'unité défaillante puissent prendre la main instantanément. Ceci conduit à imposer que chaque unité ait une capacité résiduelle au moins égale à $\frac{1-\gamma}{n-1}x$. Lorsque l'équation (2.1) est satisfaite le système est à *pleine charge*. Notons qu'en considérant des systèmes en redondance $(n-1)$ sur n nous avons tendance à considérer des systèmes qui sont un peu plus chargés qu'en pratique. Ceci dans la mesure où un système composé

de 10 unités ou plus est supposé fonctionner en redondance $(n-2)$ sur n (pour des raisons que nous ne détaillerons pas ici).

En pratique, il est rare qu'un système soit à pleine charge dans la mesure où les processus n'ont généralement pas tous la même consommation et qu'ils ne sont pas morcelables.

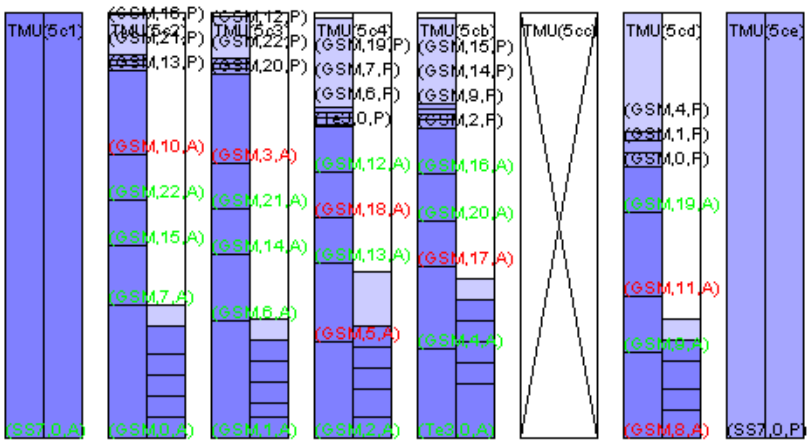
2.3.4 Sources de dégradation de l'état initial

Au démarrage, l'état du système possède de *bonnes* propriétés, en particulier l'équirépartition de la charge, ainsi qu'illustré sur la figure 2.5.

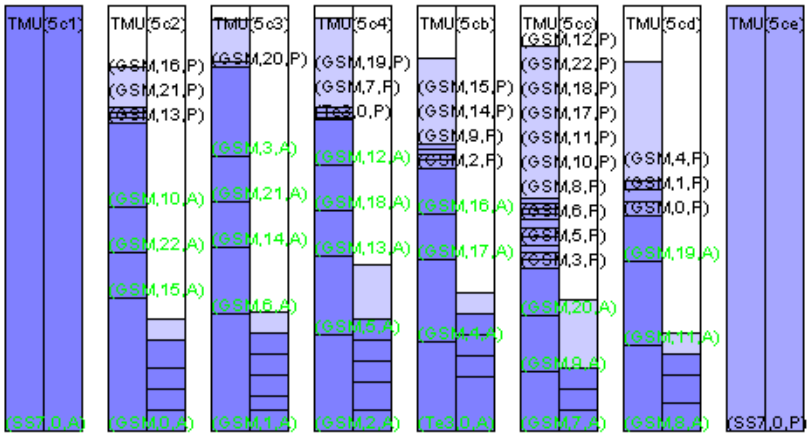
Suite à la défaillance d'une unité de traitement d'appels il est critique de stabiliser le système aussi vite que possible. Il convient en conséquence de limiter le nombre d'opérations réalisées sur la charge de traitement d'appels au strict nécessaire : basculements d'activité puis redémarrage des processus actifs perdus (s'il y en a et si possible) puis redémarrage des processus passifs manquants (s'il y en a et si possible). Des considérations identiques conduisent à limiter l'utilisation de l'opération de migration (section 2.3.1) lorsqu'une unité de traitement d'appels redémarre (nous reviendrons sur ce sujet). Par conséquent, l'état du système après la défaillance puis le redémarrage d'une unité de traitement d'appels n'est généralement pas équivalent à l'état qui précédait à cette défaillance. Il est même souvent de moins bonne qualité : la charge est moins bien équilibrée et il se peut, dans de rares, cas que certains passifs manquent à l'appel (et seulement des passifs [8]). Pour remédier à cela, il convient donc de fournir une procédure qui permet de restaurer, sans redémarrage effectif, l'état système de démarrage (ou un état qui lui serait équivalent) à partir d'un état du système arbitraire, c'est-à-dire de reconfigurer la charge du système.

Par ailleurs, l'ensemble des stations de base géré par le système peut être modifié dynamiquement : ajout d'une nouvelle station, suppression d'une station existante et ajout ou suppression de TRX dans une station existante. En plus de leur utilisation dans le cadre de l'exploitation quotidienne du sous-système radio, ces possibilités sont aussi utilisées lors de la mise en service d'un BSCe3 dans un réseau existant. Ce dernier est alors démarré à vide et les stations de base sont ajoutées dynamiquement dans un ordre arbitraire. En conséquence, l'algorithme de répartition est privé de l'information précieuse que donne la connaissance *a priori* de l'ensemble des stations à gérer. Ceci fournit une motivation supplémentaire pour vouloir restaurer un état du système équivalent à l'état de démarrage (qui, lui, est construit en exploitant la connaissance *a priori* de l'ensemble des stations de base).

8. En effet, nous avons montré (Sirdey et al., 2003) que les algorithmes utilisés garantissent que l'intégralité de la charge de traitement d'appels est dûment supportée dès qu'un nombre minimum d'unités de traitement d'appels sont opérationnelles. Ce résultat est indépendant de l'histoire du système.



(a) État du système après défaillance de la TMU 0x5cc.



(b) État du système après redémarrage de la TMU 0x5cc (à comparer avec la figure 2.5).

FIGURE 2.6 – Illustration des mécanismes de tolérance aux pannes du BSC.

De manière imagée, la procédure de reconfiguration de la charge du système peut être vue comme une opération de défragmentation : on s'autorise l'utilisation de procédures simples et robustes pour traiter les défaillances des unités de traitement d'appels ou les modifications dynamiques de l'ensemble des stations de base géré par le système et ce n'est que lorsque la qualité de l'état du système se dégrade de manière significative (néanmoins sans impact sur le service) qu'il devient recommandé d'appliquer une reconfiguration globale de la charge de traitement d'appels.

2.4 Spécification de la procédure de reconfiguration

2.4.1 Les grandes étapes

La procédure de reconfiguration est supposée n'être appliquée qu'en heure creuse.

Au plus haut niveau, elle se décompose comme suit :

1. arrêt temporaire des processus passifs ;
2. reconfiguration de la charge active ;
3. redémarrage des processus passifs.

L'intérêt de l'arrêt temporaire des processus passifs est double : cela permet de libérer des ressources, ce qui a évidemment tendance à faciliter la reconfiguration de la charge active, et cela induit un rafraîchissement des données des processus passifs. De plus, comme nous l'avons vu à la section 2.3.1, il convient de toute façon d'arrêter temporairement les passifs des actifs déplacés de manière à éviter l'apparition d'incohérences durant la reconfiguration.

2.4.2 Reconfiguration de la charge active

Les propriétés d'un état du système sont invariantes par une permutation des TMU et par une permutation de deux processus actifs de même taille à condition, et c'est très important, de permuter aussi les processus passifs correspondants. Deux TMU peuvent donc être associées afin que la première devienne équivalente à la seconde, dans l'état final, à la suite de la reconfiguration. En particulier, il convient d'enlever de la première TMU tous les processus qui ne tournent pas sur la seconde dans l'état final. Ceci nous permet donc de quantifier a priori le nombre de déplacements à réaliser. Plus formellement, soit P_u^i (respectivement $P_{u'}^f$) l'ensemble des processus qui tournent sur la TMU u (respectivement u') dans l'état initial (respectivement final) alors si u doit devenir équivalente à l'état final

de u' il convient de réaliser $|P_u^i \setminus P_{u'}^f|$ déplacements [9]. Une permutation optimale des TMU s'obtient alors par résolution du programme linéaire en nombres entiers suivant [10] :

$$\begin{cases} \text{Minimiser } \displaystyle\sum_{u\in U}\sum_{u'\in U} |P_u^i \setminus P_{u'}^f| x_{uu'}, & \\ \text{s. l. c.} & \\ \displaystyle\sum_{u\in U} x_{uu'} = 1 & \forall u' \in U, \\ \displaystyle\sum_{u'\in U} x_{uu'} = 1 & \forall u \in U, \\ x_{uu'} \in \{0,1\} & \forall u \in U, u' \in U, \end{cases}$$

qui n'est rien d'autre qu'une instance du problème d'affectation, problème polynomial bien connu (Korte & Vygen, 2000).

Cette technique de prétraitement permet de réduire le nombre de déplacements de l'ordre de 20%. Ceci est illustré sur la figure 2.7.

Notons aussi que cette approche se généralise relativement aisément afin de tenir aussi compte des simplifications induites par les permutations de processus de même taille (Sirdey, 2004a).

Nous procédons donc en deux temps : nous commençons par calculer un ensemble de déplacements de cardinal minimum puis nous l'ordonnons indépendamment. Ceci est motivé par plusieurs raisons. Sur le plan opérationnel, tout d'abord, nous l'avons vu, réaliser une migration de processus est une opération complexe. Il y a donc un risque inhérent à la réalisation d'une telle migration et minimiser le nombre de déplacements c'est aussi minimiser ce risque. Sur le plan théorique, enfin, il est plus facile de traiter les deux problèmes séparément et minimiser le nombre de déplacements c'est aussi minimiser la taille des instances du problème d'ordonnancement induit qui lui, nous allons le voir, est difficile.

Par contre, travailler en deux temps déstructure le problème : il se peut par exemple qu'il n'existe un ordonnancement sans impact sur le service que pour une permutation des TMU non optimale en termes de nombre de déplacements. Ceci étant dit, ce n'est que rarement le cas sur les instances rencontrées dans la pratique.

Le problème d'ordonnancement induit, appelé *problème de reconfiguration*, est dûment formalisé au chapitre suivant. Notons néanmoins que le calcul de l'ordonnancement pourra éventuellement se faire en arrière-plan.

9. Si l'on suppose, sans perte de généralité (cf. section 3.1.3), que tous les processus affectés à un processeur dans l'état initial le sont aussi dans l'état final.

10. On peut tout à fait imaginer d'autres critères : minimisation de la charge totale déplacée, maximisation de la charge initialement déplacable, etc.

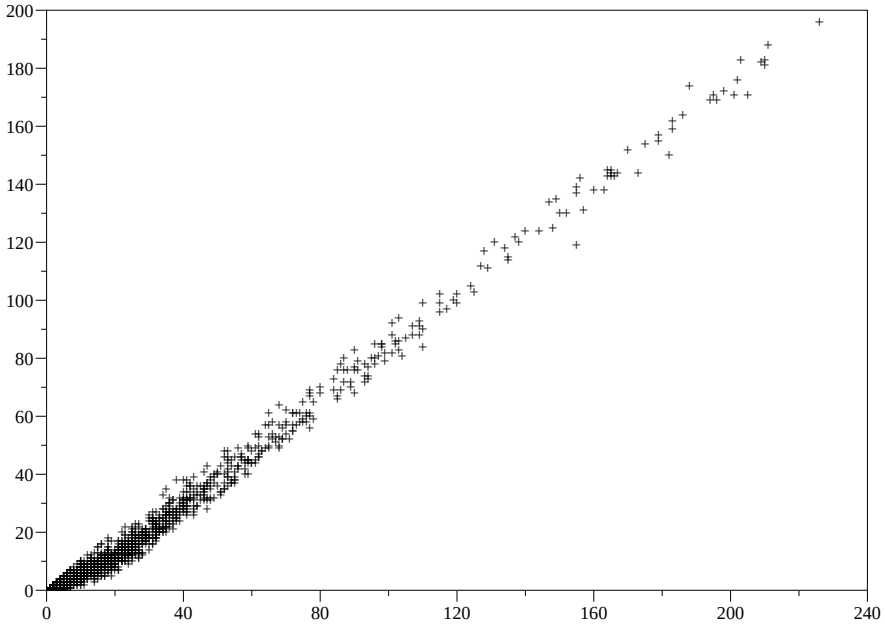


FIGURE 2.7 – Nombre de déplacements minimum en fonction du nombre de déplacements sans permutation des TMU. Sur une base d'instances aléatoires réalistes.

2.4.3 Conséquences d'une défaillance

Lorsqu'une défaillance de TMU survient en cours de reconfiguration [11], l'état final perd son sens. Il convient donc d'abandonner la reconfiguration. De plus, la reconfiguration est une opération sensible qui ne doit pas être déclenchée (ou redéclenchée) de manière spontanée. L'opérateur se trouve donc informé de l'abandon et peut décider de relancer la reconfiguration ou de la reporter à plus tard, en particulier lorsque la fenêtre de maintenance touche à sa fin. Notons, aussi, qu'il ne s'agit pas d'un abandon pur et simple : les mécanismes de traitement de pannes du BSC sont invoqués de manière à remettre le système dans un état raisonnable. En particulier, les passifs sont recréés.

11. Que ce soit pendant le calcul de l'ordonnancement (qui peut prendre un certain temps, en fonction de la méthode utilisée) ou pendant sa réalisation.

Deuxième partie

Résultats

Introduction

Cette partie résume notre contribution à l'étude et à la résolution du problème de reconfiguration.

Le chapitre 3 introduit le problème de manière formelle. Il s'agit d'un problème NP-difficile au sens fort et nous concluons le chapitre par un point de vue personnel sur les implications de ce résultat en termes de génie logiciel, domaine auquel nous avons été amené à nous intéresser en profondeur en marge de notre travail de thèse (Sirdey, 2005).

Le chapitre 4 présente nos travaux en termes de résolution exacte combinatoire. Nous y abordons donc essentiellement l'étude de cas particuliers polynomiaux du problème de reconfiguration ainsi qu'un algorithme de résolution exacte non polynomial pour le cas général. La pertinence pratique de ce dernier algorithme est illustrée par le biais de résultats expérimentaux.

Le chapitre 5 est consacré aux aspects polyédriques de nos travaux. Le lecteur y trouvera un certain nombre d'éléments théoriques sur deux polytopes que nous avons introduits : les polytopes des sous-tournois sans circuit et des programmes de déplacements. Puis, un algorithme de résolution exacte polyédrique basé sur ces résultats est présenté. Cet algorithme se distingue du précédent au sens où l'approche polyédrale nous donne des bornes inférieures globales de bonne qualité, ce qui permet de quantifier de manière assez précise l'écart à l'optimum lorsque la taille des instances augmente. Des résultats empiriques illustrent l'intérêt pratique de ce travail.

Le chapitre 6, quant à lui, résume nos travaux en termes de résolution approchée à l'aide de la méthode du recuit simulé. Nous y présentons des résultats théoriques issus de l'application de la théorie des chaînes de Markov à la méthode et, guidé par ces résultats, un algorithme pseudopolynomial de résolution approchée du problème de reconfiguration. C'est dans le contexte de l'étude expérimentale des performances de cet algorithme qu'est démontré l'intérêt de nos travaux sur la résolution exacte : notre capacité à résoudre exactement ou presque un grand nombre d'instances nous permet de vérifier de manière précise l'adéquation entre la pratique et la théorie, sur le plan du comportement de notre algorithme de recuit en termes d'écart à l'optimum.

Dans cette partie, les résultats sont énoncés sans preuve et nous nous permettons de renvoyer le lecteur aux articles que nous avons publiés dans la littérature

spécialisée pour plus de détails. L'introduction de chacun des chapitres de cette partie indique le ou les articles auxquels il convient de se reporter.

Chapitre 3

Formalisation du problème de reconfiguration, complexité

Introduction

Le problème de reconfiguration, dont la résolution fait l'objet de ce mémoire, est un problème d'ordonnancement NP-difficile au sens fort à contraintes de ressource.

Comme nous l'avons vu au chapitre précédent, il s'agit d'ordonner des déplacements de processus de traitement d'appels, qui consomment les ressources offertes par les processeurs d'un autocommutateur réparti (erlangs et ports de communication, notamment), de telle manière que des contraintes de capacité sur ces derniers ne soient jamais violées. Les situations de blocage, parfois inévitables, sont alors résolues en arrêtant temporairement des processus, donc en renonçant temporairement à rendre une partie du service, l'objectif étant d'avoir le moins possible recours à ce mécanisme.

La figure 3.1 représente une instance du problème de reconfiguration pour un système à 10 processeurs, une ressource et 46 processus. La capacité de chacun des processeurs dans l'unique ressource est égale à 30 et la somme des consommations des processus est égale à 281. Les figures d'en haut et d'en bas représentent respectivement l'état initial et l'état final. Par exemple, le processus 23 doit être déplacé du processeur 2 vers le processeur 6.

Ce chapitre est consacré à la formalisation du problème de reconfiguration ainsi qu'à l'étude de sa complexité.

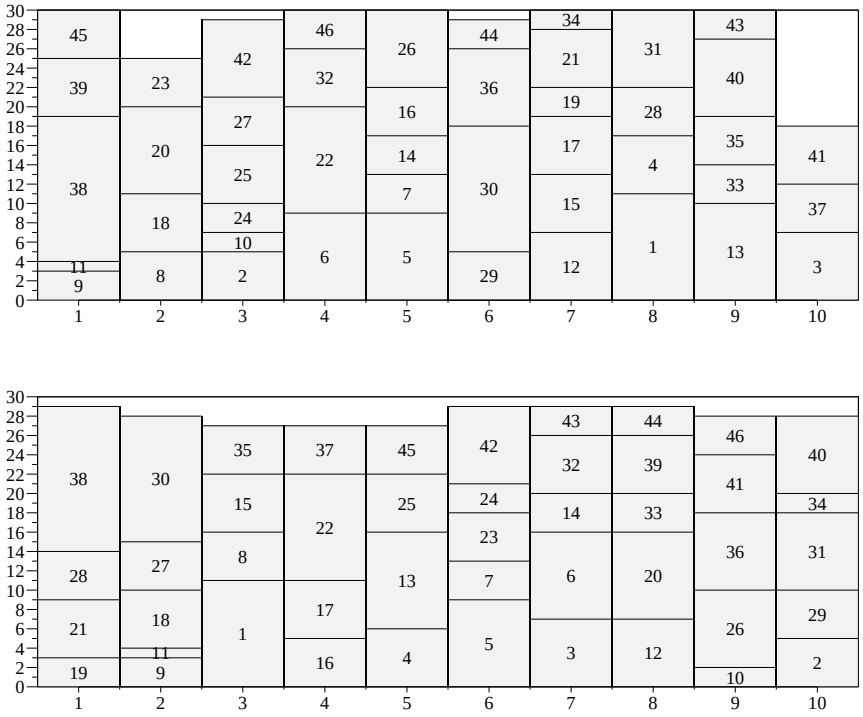


FIGURE 3.1 – Exemple d'instance du problème de reconfiguration.

3.1 Énoncé du problème

3.1.1 Notion de système réparti

Nous énonçons le problème de reconfiguration dans le cadre très large des systèmes répartis (Lynch, 1996).

Un *système réparti* est défini comme une entité composée d'un certain nombre de *processeurs* ou *sites* interconnectés par un certain nombre de *liens*. Cette définition est volontairement très large : elle englobe les réseaux de télécommunications à grande distance, les réseaux locaux ainsi que les machines parallèles à mémoire répartie, dont notre autocommutateur est un exemple.

L'*algorithmique répartie*, qui traite de la conception d'algorithmes *prouvés* fonctionner correctement dans un contexte où les processeurs et les liens opèrent potentiellement à des vitesses différentes et où certains composants peuvent tomber en panne à tout instant, est un domaine extrêmement riche de l'informatique théorique.

Néanmoins, il nous semble opportun d'indiquer qu'en dépit du contexte le problème de reconfiguration n'est pas un problème d'algorithmique répartie. Ceci dans la mesure où nous faisons l'hypothèse de l'existence d'une entité coordinatrice, supposée parfaitement fiable, extérieure au système à reconfigurer. Enfin, comme nous l'avons vu à la section 2.4.3, le problème des pannes est éludé au sens où la défaillance d'un ou plusieurs des processeurs du système se traduit par l'abandon pur et simple de la reconfiguration en cours, l'état final perdant généralement son sens.

3.1.2 Premières notations

Soit donc un système réparti composé d'un ensemble U de processeurs et soit R l'ensemble des ressources qu'ils fournissent (CPU, erlangs, ports, etc.). Pour chaque processeur $u \in U$ et chaque ressource $r \in R$, $C_{u,r} \in \mathbb{N}$ indique la quantité de la ressource r offerte par le processeur u.

Soit P un ensemble d'applications, appelées *processus* par la suite, qui ont vocation à consommer les ressources offertes par les processeurs. L'ensemble P s'appelle aussi la *charge utile* du système. Étant donné un processus $p \in P$ et une ressource $r \in R$, $w_{p,r} \in \mathbb{N}$ dénote la quantité de la ressource r consommée par le processus p.

Notons par ailleurs que ni $C_{u,r}$ ni $w_{p,r}$ ne sont supposés varier au cours du temps.

Enfin, dans le cas à une ressource, nous omettons le r en indice.

3.1.3 Notion d'état admissible

Un *état système admissible* est une fonction $f : P \longrightarrow U \cup \{u_\infty\}$, où u_∞ est un processeur fictif de capacité infinie, telle que pour tout $u \in U$ et tout $r \in R$ on a

$$\sum_{p \in P(u;f)} w_{p,r} \leq C_{u,r}, \tag{3.1}$$

avec $P(u; f) = \{p \in P : f(p) = u\}$.

Les processus appartenant à $\bar{P}(f) = P(u_\infty; f)$ ne sont affectés à aucun processeur. Lorsque cet ensemble n'est pas vide, on dit que le système fonctionne en *mode dégradé*.

Une instance du problème de reconfiguration est alors entièrement spécifiée par la donnée de deux états admissibles arbitraires, l'*état initial*, noté f_i et l'*état final*, noté f_t.

Notons, de plus, que nous faisons l'hypothèse que $\bar{P}(f_i) = \bar{P}(f_t) = \emptyset$. Lorsque cela s'avère ne pas être le cas, alors il convient de commencer par arrêter les processus appartenant à $\bar{P}(f_t) \setminus \bar{P}(f_i)$, puis de procéder à la reconfiguration et, enfin, de démarrer les processus de $\bar{P}(f_i) \setminus \bar{P}(f_t)$. Les processus de $\bar{P}(f_i) \cap \bar{P}(f_t)$, quant à eux, n'ont aucune pertinence.

3.1.4 Déplacement de processus

Un processus peut être déplacé d'un processeur vers un autre de deux manières : soit il est *migré*, auquel cas il consomme des ressources sur les deux processeurs pendant l'intégralité du déplacement qui n'a pas d'impact sur le service, soit il est *interrompu*, c'est-à-dire arrêté puis ultérieurement redémarré sur le processeur cible du déplacement. Bien évidemment, cette dernière opération induit une perte de service temporaire.

Par ailleurs, il convient de garantir que les contraintes de capacité (3.1) sont respectées pendant la reconfiguration ou, autrement dit, que tous les états intermédiaires sont admissibles, et que chaque processus n'est déplacé (migré ou interrompu) qu'une seule fois au plus. Cette dernière contrainte découle du fait qu'un déplacement de processus, une migration en particulier, est une opération complexe et potentiellement risquée. En conséquence, il convient de se limiter au strict nécessaire.

Quelque peu abusivement, nous disons qu'un déplacement est *interrompu* lorsqu'il est réalisé par interruption du processus concerné. Ceci allège significativement notre discours. De plus, nous nous plaçons maintenant dans le cas à une ressource (*i. e.*, $|R| = 1$), à moins d'indication contraire.

3.1.5 Notion de programme de déplacement

Soit $u \in U$, un processus p appartenant à $P(u; f_i) \setminus P(u; f_t)$ doit être déplacé de u à $f_t(p)$. Soit donc M l'ensemble des déplacements. Pour chaque $m \in M$, w_m, s_m et t_m dénotent respectivement la quantité de ressource déplacée, le processeur source du déplacement et le processeur cible du déplacement. Enfin, on note $S(u) = \{m \in M : s_m = u\}$, respectivement $T(u) = \{m \in M : t_m = u\}$, les ensembles de processus à enlever du, respectivement à mettre sur le, processeur u.

Un couple (I, σ), où $I \subseteq M$ et où $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ est une bijection, définit un *programme de déplacement* si, sous l'hypothèse que les déplacements de I sont interrompus, on peut migrer les déplacements de $M \setminus I$ dans l'ordre spécifié par σ sans induire de violation des contraintes de capacité. Plus formellement, (I, σ) est un programme de déplacement si pour tout $m \in M \setminus I$ on a

$$w_m \leq K_{t_m} + \sum_{\substack{m' \in I \\ s_{m'} = t_m}} w_{m'} + \sum_{\substack{m' \in S(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'}, \tag{3.2}$$

avec $K_u = C_u - \sum_{p \in P(u; f_i)} w_p$.

Par ailleurs, remarquons que dans la mesure où l'état final est admissible on a, pour chaque processeur $u \in U$

$$K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u)} w_m \geq 0. \tag{3.3}$$

Soit c_m le coût associé à l'interruption du déplacement m. Le problème de reconfiguration consiste, étant donné un ensemble M de déplacements, à trouver un couple (I, σ) tel que $c(I) = \sum_{m \in I} c_m$ soit minimum.

3.2 État de l'art

Nous nous permettons de renvoyer le lecteur à l'étude bibliographique de la section 2 (page 106) de notre article paru dans le *European Journal of Operational Research*, reproduit page 103.

3.3 Complexité

3.3.1 *NP*-complétude au sens fort

L'un des tout premiers résultats que nous avons obtenu concerne la complexité du problème de reconfiguration. En effet, nous avons montré que ce problème est NP-difficile au sens fort. La preuve de ce résultat, par restriction du cas à deux

processeurs au problème 3-partition, est donnée dans notre article à paraître dans le *European Journal of Operational Research*, reproduit page 103.

Ce résultat est fondamental dans la mesure où il permet d'orienter notre recherche vers certaines techniques algorithmiques plutôt que vers d'autres : à moins que la fameuse conjecture $P \neq NP$ soit fausse, il n'existe ni d'algorithme polynomial ni d'algorithme pseudopolynomial capable de résoudre exactement le problème de reconfiguration.

3.3.2 Approches algorithmiques

Comme nous l'avons vu au chapitre 1, le caractère intrinséquement difficile du problème de reconfiguration justifie les approches suivantes :

1. chercher s'il existe des cas particuliers du problème qu'il est possible de résoudre exactement à l'aide d'algorithmes efficaces (chapitre 4) ;
2. chercher à résoudre le problème de manière exacte à l'aide d'algorithmes non polynomiaux (chapitres 4 et 5).
3. chercher à résoudre le problème de manière approchée à l'aide d'algorithmes efficaces (chapitre 6) ;

Laquelle de ces approches est la plus appropriée ? Comme souvent en ingénierie il n'y a pas de réponse toute faite.

Du logiciel opérationnel se doit d'être *maintenable*, *fiable* et *efficace* (Sirdey, 2005). Généralement, la maintenabilité prime car un logiciel est continuellement modifié, que ce soit pour introduire de nouvelles fonctionnalités, corriger des erreurs ou améliorer son efficacité. En conséquence, un logiciel fiable à l'instant t risque fort de ne pas le rester très longtemps s'il n'est pas maintenable. Enfin, la fiabilité prime généralement sur l'efficacité car contrairement aux erreurs qui tendent à être dispersées un peu partout dans le logiciel, les sources d'inefficacité sont généralement très localisées et peuvent (doivent) être identifiées rationnellement à l'aide d'outils appropriés (Knuth, 1974 ; Fowler, 2004).

Comment ceci se traduit-il lorsqu'il s'agit de choisir une stratégie de résolution d'un problème combinatoire difficile ? D'emblée, à l'exception des cas où « *even small percentage savings amount to substantial dollar amounts* » (Hoffman & Padberg, 1993), le critère de maintenabilité conduit à éliminer les approches requérant un degré trop important de raffinement mathématique (pour se fixer les idées, un document d'une cinquantaine de pages au plus doit suffire à véhiculer le fonctionnement de la méthode) ou introduisant un trop grand nombre de lignes de code (typiquement, de l'ordre du millier de lignes, sans compter les librairies, pour résoudre un problème isolé). Les personnels changent, ce qui est normal (et même souhaitable dans la limite du raisonnable), et l'organisation responsable doit être capable de garder la maîtrise du logiciel. Ce dernier doit donc être appréhendable à coût raisonnable par un ingénieur, afin de pouvoir le faire évoluer ou corriger les éventuelles erreurs résiduelles sans en dégrader la fiabilité.

Il convient donc, et c'est plus facile à dire qu'à faire, de trouver le bon compromis entre la complexité du logiciel et la qualité des solutions qu'il produit.

L'efficacité est sujette à encore plus de discussions. S'il paraît raisonnable, dans le cadre d'une opération de maintenance planifiée d'un grand système (par exemple un réseau de télécommunications ou une centrale nucléaire), d'avoir recourt à l'approche exacte et de dédier quelques heures, voire quelques jours, de calculs à la recherche d'une solution optimale, cela n'est pas le cas pour du logiciel embarqué soumis à des contraintes temps réel. Dans ce dernier cas, il convient de se tourner vers des méthodes *efficaces* de résolution approchée sans oublier, en l'absence de garanties théoriques acceptables, d'évaluer leurs performances à l'aide de méthodes exactes.

Sans prétendre à une quelconque universalité, ceci résume la démarche que nous avons suivie [1].

1. Le lecteur peut se reporter au chapitre 7 de l'ouvrage de Hromkovič (2003) pour une discussion particulièrement pragmatique autour du même thème.

Chapitre 4

Approche combinatoire

Introduction

Les travaux résumés dans ce chapitre font l'objet d'un article paru en 2007 dans le *European Journal of Operational Research* (volume 183, pages 546 à 563).

Nous abordons ici la résolution exacte du problème de reconfiguration sous l'angle combinatoire, c'est-à-dire sous l'angle de l'exploitation directe de la structure du problème (ceci en opposition avec l'approche polyédrale qui est le sujet du chapitre 5). Nous commençons par introduire les cas particuliers du problème pour lesquels nous avons pu trouver des algorithmes polynomiaux. Nous présentons ensuite un algorithme de résolution exacte par recherche arborescente, cette présentation s'accompagne de résultats empiriques qui illustrent la pertinence pratique de l'algorithme.

4.1 Cas particuliers polynomiaux

L'étude des cas particuliers polynomiaux présentés dans cette section repose sur les notions de graphe et de multigraphe de transfert que nous avons introduites. Le *multigraphe de transfert* est défini comme le multigraphe [1] orienté dont les sommets sont les processeurs et les arcs les déplacements. Le *graphe de transfert*, quant à lui, s'obtient à partir du multigraphe de transfert en contractant les arcs parallèles.

4.1.1 Graphes de transfert sans circuit

Lorsque le graphe de transfert, et *a fortiori* le multigraphe de transfert, ne contient pas de circuit, nous avons montré que les déplacements peuvent toujours être ordonnés de manière à ce qu'aucune interruption ne soit nécessaire. Un tel

1. Les arcs parallèles sont autorisés, les boucles ne le sont pas (Bang-Jensen & Gutin, 2002).

ordonnancement se déduit aisément d'une *bonne numérotation* des sommets du graphe c'est-à-dire d'une fonction f de l'ensemble des sommets vers l'ensemble des entiers qui vérifie $f(v) < f(w)$ pour tout arc (v, w) appartenant au graphe. Une telle numérotation peut être obtenue en temps linéaire.

La figure 4.1 illustre le principe de fonctionnement de la méthode. Dans la mesure où l'état final est admissible et puisqu'il n'y a rien à enlever du processeur 4, le dernier selon une bonne numérotation, tous les déplacements qui ciblent ce processeur sont réalisables. On peut donc réaliser les deux déplacements de 3 vers 4 et de 8 vers 4, ce qui a pour effet de garantir que le déplacement de 8 vers 3 est réalisable, dans la mesure où il ne reste plus rien à enlever du processeur 3. C'est ensuite du processeur 8 qu'il n'y a plus rien à enlever, et ainsi de suite.

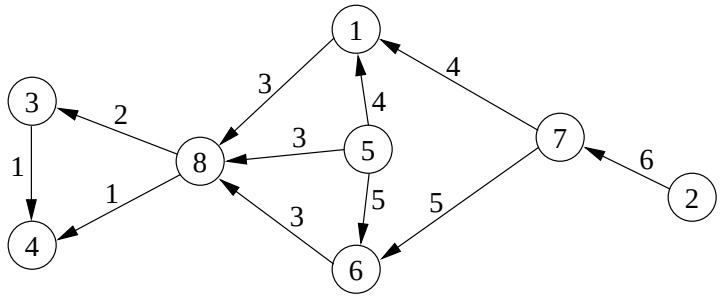


FIGURE 4.1 – Résolution du cas particulier où le graphe de transfert est sans circuit.

Lorsque le multigraphe de transfert contient des circuits, nous avons montré que ses composantes fortement connexes peuvent être traitées indépendamment, selon un ordre qui se déduit d'une bonne numérotation des sommets du graphe de transfert réduit (graphe obtenu en contractant les composantes fortement connexes). Ceci permet de décomposer le problème en sous-problèmes plus simples, sous certaines conditions (multigraphe de transfert non fortement connexe).

La figure 4.2 illustre ce principe de décomposition. On commence comme dans le cas sans circuit : tous les déplacements qui ciblent le processeur 2 (10 vers 2 et 12 vers 2) sont réalisables, par admissibilité de l'état final. Les réaliser permet de garantir que le déplacement de 12 vers 10 devient réalisable. Réaliser ces déplacements vers des sommets situés en aval de la composante fortement connexe A avant de se préoccuper des déplacements internes à cette composante ne peut être que bénéfique dans la mesure où des ressources sont libérées sur les processeurs de A. De la même manière, remarquons que réaliser des déplacements dont la cible est dans A et dont la source n'appartient pas à A (4 vers 5 et 4 vers 6) avant d'en avoir terminé avec les déplacements internes à A ne présente aucun intérêt. En effet, ces déplacements seront nécessairement réalisables une fois que les déplacements dont la source est soit dans A soit en aval de A auront été réalisés, par admissibilité de l'état final. Les réaliser prématurément ne peut donc que faire peser des contraintes inutiles sur les ordonnancements admissibles

des déplacements internes à A.

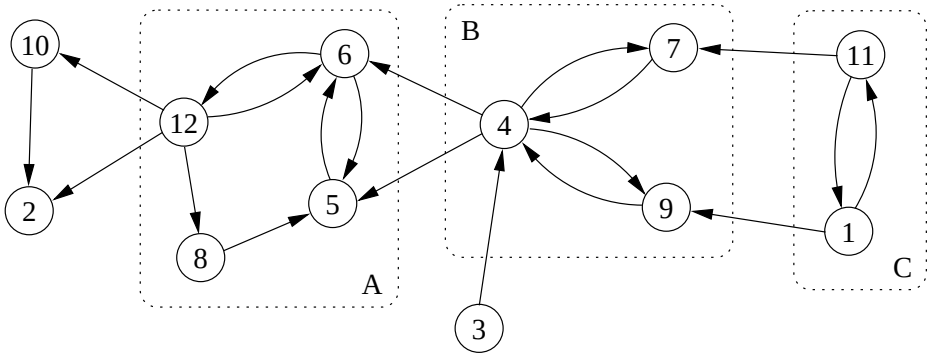


FIGURE 4.2 – Décomposition par analyse de la connexité forte.

4.1.2 Cas homogène

Lorsque les processeurs n'offrent qu'un seul type de ressources et lorsque tous les processus ont la même consommation, nous avons montré que le problème de reconfiguration peut être résolu de manière efficace. Pour ce faire, nous avons proposé un algorithme, démontré sa validité et prouvé que sa complexité est bien polynomiale. Succinctement, cet algorithme procède itérativement par extraction d'un sous-graphe eulérien maximal de la dernière (au sens d'une bonne numérotation) composante fortement connexe du multigraphe de transfert résiduel, les déplacements du sous-graphe étant réalisés dans l'ordre donné par un circuit eulérien d'origine bien choisie.

La figure 4.3 illustre le fonctionnement de cet algorithme. Initialement, le graphe de transfert est fortement connexe. On choisit alors un sous-graphe eulérien maximal passant par le sommet 2. Ce sous-graphe est représenté en pointillé sur la figure. Dans la mesure où le processeur 2 est la cible de deux déplacements et la source d'un seul, l'admissibilité de l'état final implique qu'au moins un déplacement vers ce processeur est réalisable. On peut donc l'utiliser comme origine pour réaliser les déplacements du sous-graphe, dans le sens inverse d'un circuit eulérien : 1 vers 2 puis 6 vers 1 puis ... puis 3 vers 4, 1 vers 3, 3 vers 1 et enfin 2 vers 3. Dès lors, le graphe n'a plus que 2 composantes connexes, $\{2, 3\}$ et $\{4, 5, 7, 8\}$, qui peuvent être considérées indépendamment. La composante $\{4, 5, 7, 8\}$ possède trois composantes fortement connexes ($\{4\}$, $\{7, 8\}$ and $\{5\}$, selon une bonne numérotation). On commence donc par réaliser le déplacement de 8 vers 5, ceci libère une unité sur 8 et permet d'utiliser ce sommet pour initier le parcours du petit sous-graphe eulérien maximal [2] (7 vers 8 puis 8 vers 7). Le graphe résiduel est sans circuit, nous avons donc terminé.

2. Ici, 7 aurait aussi pu être utilisé dans la mesure où ce processeur est la cible de deux déplacements et n'est la source que d'un, par admissibilité de l'état final donc.

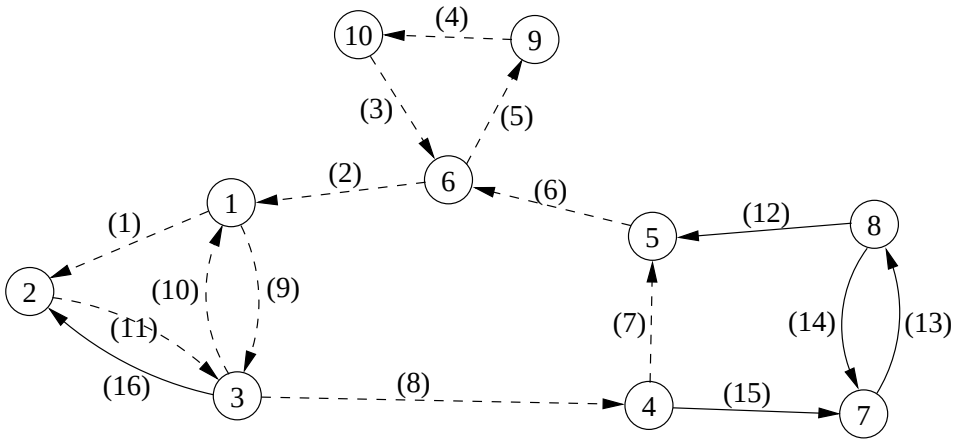


FIGURE 4.3 – Résolution du cas homogène.

Aussi, nous avons pu caractériser la valeur des solutions optimales[3] : si le graphe de transfert est eulérien (tout sommet du graphe est tel que ses demi-degrés intérieur et extérieur sont égaux) et si aucun déplacement n'est initialement réalisable alors il est nécessaire et il suffit d'interrompre le processus associé à un déplacement dont le coût est égal à $\min_{m \in M} c_m$. Dans le cas contraire, les déplacements peuvent toujours être ordonnés de manière à ce qu'aucune interruption ne soit requise et notre algorithme trouve justement un tel ordonnancement.

Enfin, nous avons identifié des conditions suffisantes sur la capacité initiale des processeurs pour que le cas général soit réductible au cas homogène. Néanmoins ces conditions sont assez restrictives et sont donc d'un intérêt pratique assez limité.

4.2 Résolution exacte combinatoire

Nous présentons dans cette section un algorithme de recherche arborescente combinatoire (*branch-and-bound*) pour le problème de reconfiguration.

4.2.1 Schéma de branchement

Au lieu de chercher, à partir de l'état initial, à atteindre l'état final, ce qui est la première idée naturelle, il s'avère plus fructueux de chercher à améliorer la pire des solutions (celle qui consiste à interrompre tous les déplacements), en évitant certaines interruptions.

Selon ce point de vue, un sommet de l'arborescence est défini par un quadruplet (I, J, σ_J, R) où I, J et R dénotent respectivement les ensembles des

3. Sous l'hypothèse que le multigraphe de transfert est simplement connexe. Si tel n'est pas le cas cette caractérisation est valable séparément pour chacune des composantes connexes.

déplacements définitivement interrompus, non interrompus et non définitivement interrompus, σ_J étant un ordonnancement des déplacements de J. En particulier, il est requis que le programme de déplacement $(I \cup R, \sigma_J)$ soit réalisable.

La racine de l'arborescence correspond au sommet $(\emptyset, \emptyset, \sigma_\emptyset, M)$.

Notre schéma de branchement consiste à chercher à concaténer des déplacements de R à σ_J tout en préservant l'admissibilité de ce dernier. Une telle concaténation est possible si le processus associé au déplacement $m \in R$ peut rester sur sa source durant l'intégralité de l'exécution de σ_J. Les processus associés aux déplacements de I et de R étant interrompus, il est alors garanti que le processeur cible de m a suffisamment de capacité pour accueillir le processus associé à m. Une feuille est obtenue lorsque $R = \emptyset$.

Ce schéma de branchement est complété par une borne inférieure et des règles de dominance.

4.2.2 Bornes inférieures

Pour un nœud N donné de l'arborescence, notons $\ell_u(N)$ la plus petite capacité résiduelle sur le processeur u pendant l'exécution du programme $(I \cup R, \sigma_J)$. Il est alors clair que $\ell_u(N)$ limite la somme des consommations des déplacements de R que l'on va pouvoir éviter d'interrompre.

Il suit que

$$\sum_{m \in I} c_m - \sum_{u \in U} \mathrm{KP}(u) - \sum_{m \in S(u) \cap R} c_m,$$

où $\mathrm{KP}(u)$ dénote la valeur d'une solution optimale du problème de sac à dos

$$\begin{cases} \mathrm{KP}(u) = \text{Maximiser} \displaystyle\sum_{m \in S(u) \cap R} c_m x_m, \\ \text{s. l. c.} \\ \displaystyle\sum_{m \in S(u) \cap R} w_m x_m \leq \ell_u(N), \\ x_m \in \{0, 1\},\ m \in S(u) \cap R, \end{cases}$$

fournit une borne inférieure sur la valeur des solutions de la branche engendrée par (I, J, σ_J, R).

Cette borne peut être calculée en temps pseudopolynomial et éventuellement déclinée en variantes plus faibles calculables en temps polynomial. Dans une certaine mesure, elle peut être généralisée au cas multiressource.

4.2.3 Règle de dominance

Aussi, nous avons introduit une règle de dominance assez générale.

Reprenons les notations de la section précédente et considérons deux nœuds de l'arborescence : $N_1 = (I_1, J_1, \sigma_{J_1}, R_1)$ et $N_2 = (I_2, J_2, \sigma_{J_2}, R_2)$. Si les conditions suivantes sont vérifiées, alors N_1 domine N_2,

1. $R_1 = R_2 = R$;
2. $\sum_{m \in I_1} c_m \leq \sum_{m \in I_2} c_m$;
3. $L_u(N_1) \geq L_u(N_2)$, $\forall u \in U$;
4. $\ell_u(N_1) \geq \ell_u(N_2)$, $\forall u \in U$;

où $L_u(N_1)$, par exemple, dénote la capacité résiduelle du processeur u après exécution du programme $(I_1 \cup R_1, \sigma_{J_1})$. Autrement dit, si l'on a exploré (ou si l'on va explorer [4]) la branche engendrée par N_1, rien ne sert d'explorer celle engendrée par N_2.

Cette règle est assez intuitive : on sent bien que si les enjeux sont les mêmes (condition 1), que si N_1 est au moins aussi bon que N_2 jusqu'à présent (condition 2) et que si N_1 est moins contraignant que N_2 sur le plan de la capacité (conditions 3 et 4) alors on ne peut pas faire moins bien en explorant la branche engendrée par N_1. Bien entendu, nous avons établi ceci de manière rigoureuse.

L'exploitation de cette règle permet l'élimination de plusieurs sources importantes de redondance. L'une d'entre elles, par exemple, concerne les nœuds qui satisfont la condition 1 et qui induisent les mêmes ordonnancements restreints pour chacun des processeurs. Dans ce cas, on a $\sum_{m \in I_1} c_m = \sum_{m \in I_2} c_m$, $L_u(N_1) = L_u(N_2)$ et $\ell_u(N_1) = \ell_u(N_2)$, pour tout $u \in U$, et, en conséquence, la règle s'applique.

4.2.4 Résultats expérimentaux

Le passage à la pratique s'est fait par le biais d'un algorithme de recherche arborescente par séparation et évaluation (*branch-and-bound*) en profondeur d'abord dont les principaux ingrédients sont la borne inférieure de la section 4.2.2 et la règle de dominance de la section 4.2.3.

Les problèmes de sac à dos qui interviennent dans le calcul de la borne inférieure sont résolus à l'aide de l'algorithme de Bellman (Kellerer et al., 2004). La dominance, quant à elle, est détectée par mémorisation de certains attributs des nœuds explorés dans une structure de données à temps d'accès logarithmique, dûment simplifiée à la volée.

Nous avons illustré la pertinence pratique de cet algorithme en l'utilisant pour attaquer des instances aléatoires *difficiles* [5]. En particulier, nous avons pu résoudre exactement des instances ayant jusqu'à 190 déplacements en moins de vingt minutes. En pratique néanmoins, si le temps de calcul est limité à vingt

4. Selon la façon d'exploiter la règle, voir la discussion dans Sirdey et al. (2007).
5. À la fois en termes de taille mais aussi de capacité résiduelle des processeurs.

minutes, l'algorithme n'a de bonnes chances de succès que jusqu'à de l'ordre de 50 déplacements.

Chapitre 5

Approche polyédrale

Introduction

Les travaux résumés dans ce chapitre font l'objet d'une série de deux rapports de recherche théoriques (rapports techniques Nortel GSM Access R&D PE/BSC/INF/017912 et PE/BSC/INF/017913), reproduits pages 149 et 171, ainsi que d'un article plus appliqué paru en 2007 dans un numéro spécial *Polyhedra and Combinatorial Optimization* du journal *RAIRO Operations Research* (volume 41, pages 235 à 251).

Nous concentrons donc notre propos sur les aspects théoriques, études des polytopes des sous-tournois sans circuit et des programmes de déplacements que nous avons introduits, et pratiques, mise en œuvre et évaluation empirique d'un algorithme de recherche arborescente polyédrique, de l'application des méthodes polyédriques à notre problème.

Afin d'éviter toute confusion avec la notion de sommet pour un polytope, nous utilisons le terme *nœud* pour désigner le sommet d'un graphe. Cette convention ne s'applique qu'au présent chapitre.

5.1 Un programme linéaire en nombres entiers

Commençons par rappeler quelques notations.

Soit U l'ensemble des processeurs du système. On note K_u la capacité résiduelle du processeur $u \in U$ dans l'état initial. Aussi, $S(u)$ et $T(u)$ dénotent l'ensemble des déplacements qui ont le processeur $u \in U$ pour source et l'ensemble de ceux qui l'ont pour cible, respectivement. Étant donné un déplacement $m \in M$, on note w_m la consommation du processus déplacé et s_m et t_m les processeurs source et cible du déplacement, respectivement.

5.1.1 Existence d'un programme de valeur nulle

Pour chaque paire $m, m' \in M$ $(m \neq m')$, nous introduisons les variables binaires

$$\delta_{mm'} = \begin{cases} 1 & \text{si } m \text{ précède à } m', \\ 0 & \text{sinon.} \end{cases}$$

En plus de satisfaire les contraintes (Queyranne & Schulz, 1994)

$$\begin{cases} \delta_{mm'} + \delta_{m'm} = 1 & \forall \{m, m'\} \subseteq M, \quad (5.1) \\ \delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M \text{ (transitivité)}, \quad (5.2) \end{cases}$$

un programme de déplacement admissible de valeur nulle se doit de satisfaire, pour tout $m \in M$, les contraintes

$$w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} \delta_{m'm} - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} \qquad \forall m \in M. \quad (5.3)$$

Ces dernières contraintes imposent qu'au moment de la réalisation du déplacement m, la capacité du processeur cible doit être suffisante pour accueillir le processus déplacé.

5.1.2 Formulation du problème de reconfiguration

Bien entendu, il se peut que le système composé des inégalités (5.1), (5.2), (5.3) et $\delta_{mm'} \in \{0, 1\}$ $(m, m' \in M,\ m \neq m')$ n'ait pas de solution, autrement dit que des interruptions soient nécessaires. En conséquence, nous introduisons, pour tout $m \in M$, les variables binaires

$$\delta_{mm} = \begin{cases} 1 & \text{si } m \text{ est interrompu,} \\ 0 & \text{sinon.} \end{cases}$$

Dans la mesure où les interruptions sont réalisées au début, il est requis que seuls les déplacements non interrompus soient ordonnés. Pour ce faire, nous avons montré (Sirdey & Kerivin, 2006c) qu'il convient de remplacer les contraintes (5.1) par les contraintes

$$\begin{cases} \delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall \{m, m'\} \subseteq M, \quad (5.4) \\ \delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M. \quad (5.5) \end{cases}$$

Enfin, en termes de capacité, les contraintes (5.3) deviennent, pour tout $m \in M$,

$$(1 - \delta_{mm}) w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} (\delta_{m'm'} + \delta_{m'm}) - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm}. \quad (5.6)$$

Le problème de reconfiguration consiste alors à minimiser

$$\sum_{m \in M} c_m \delta_{mm}$$

sous les contraintes (5.4), (5.5), (5.2), (5.6) et $\delta_{mm'} \in \{0,1\}$, pour tout $m, m' \in M$. Le polytope associé à ce programme linéaire en nombre entiers s'appelle le *polytope des programmes de déplacement* et se note P^M_{PMP}.

5.2 Sur le polytope des sous-tournois sans circuit

5.2.1 Définition

Le polytope des sous-tournois sans circuit (*partial linear ordering polytope* en anglais), noté P^n_{PLO}, est défini comme l'enveloppe convexe des vecteurs caractéristiques des ordonnancements linéaires d'un sous-ensemble des nœuds du graphe orienté complet à n sommets, les nœuds exclus de l'ordonnancement étant munis d'une boucle. La figure 5.1 fournit un exemple de point entier de P^6_{PLO}.

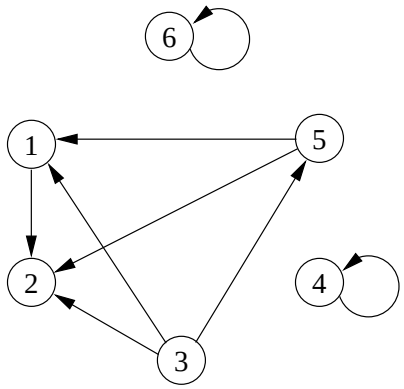


FIGURE 5.1 – Un point de P^6_{PLO}.

De manière équivalente, ce polytope correspond à l'enveloppe entière des solutions du système d'inégalités suivant

$$\begin{cases} \delta_{ij} + \delta_{ji} + \delta_{ii} + \delta_{jj} \geq 1 & 1 \leq i < j \leq n & (5.7) \\ \delta_{ij} + \delta_{ji} + \delta_{ii} \leq 1 & i,j \in \{1,\ldots,n\}, i \neq j & (5.8) \\ \delta_{ij} + \delta_{jk} - \delta_{ik} \leq 1 & i,j,k \in \{1,\ldots,n\}, i \neq j \neq k \neq i & (5.9) \\ 0 \leq \delta_{ij} \leq 1 & i,j \in \{1,\ldots,n\}, i \neq j & \end{cases}$$

Lorsque $\delta_{ij} = 1$ $(i \neq j)$ le nœud i est ordonné avant le nœud j et lorsque $\delta_{ii} = 1$ le nœud i n'est pas ordonné.

Le polytope des sous-tournois sans circuit s'obtient donc à partir du polytope des programmes de déplacement en relâchant les contraintes de capacité (5.6).

5.2.2 Propriétés élémentaires

Nous avons montré que P^n_{PLO} est de pleine dimension et que les inégalités $\delta_{ij} \geq 0$ $(i \neq j)$ ainsi que les inégalités de type (5.7) et (5.8) en définissent toujours des facettes.

Les inégalités $\delta_{ij} \geq 0$ $(i = j)$, $\delta_{ij} \leq 1$ ainsi que les contraintes de transitivité (5.9), quant à elles, ne définissent jamais des facettes de P^n_{PLO}. Les contraintes de transitivité peuvent néanmoins être remplacées par les inégalités

$$\delta_{ij} + \delta_{jk} - \delta_{ik} + \delta_{jj} \leq 1 \qquad i, j, k \in \{1, \ldots, n\}, i \neq j \neq k \neq i$$

appelées *contraintes de transitivité étendues*, qui, elles, fournissent des facettes de P^n_{PLO}.

5.2.3 Classes de facettes non triviales

Dans la mesure où optimiser sur P^n_{PLO} permet de résoudre le problème de l'ordonnancement linéaire de plus fort poids, cf. Reinelt (1985), il est peu vraisemblable qu'une description complète de P^n_{PLO} puisse être jamais obtenue. Néanmoins, nous avons mis en évidence plusieurs classes de facettes non triviales.

Soit $I \subseteq \{1, \ldots, n\}$ avec $|I| = k \geq 2$, l'inégalité

$$\sum_{i \in I} \sum_{j \in I} \delta_{ij} \geq |I| - 1 \tag{5.10}$$

est valide pour P^n_{PLO} et en définit une facette. Une telle inégalité, illustrée sur la figure 5.2, s'appelle une *k-clique*. Les k-cliques généralisent les inégalités de type (5.7), nous désignerons donc ces dernières inégalités, dans la suite, sous le vocable de *minicliques*. Pour k fixé, les k-cliques sont séparables en temps polynomial, car polynomiales en nombre. Le cas général est par contre NP-difficile au sens fort par restriction au problème de la coupe maximum.

Soit $I \subset \{1, \ldots, n\}$ avec $|I| = k$ et $i_0 \in \{1, \ldots, n\} \setminus I$, l'inégalité suivante est valide pour P^n_{PLO} et en fournit une facette :

$$\delta_{i_0 i_0} + \sum_{i \in I} (\delta_{i i_0} + \delta_{i_0 i}) - \sum_{i \in I} \sum_{j \in I \setminus \{i\}} \delta_{ij} \leq 1. \tag{5.11}$$

Ces inégalités, qui généralisent les inégalités de type (5.8) et dont la structure du membre gauche est illustrée sur la figure 5.3, s'appellent des *k-monocycles*. Les inégalités de type (5.8) sont donc baptisées *minimonocycles*. De nouveau, pour k fixé, les k-monocycles sont séparables en temps polynomial, car polynomiales en nombre. Le cas général est par contre NP-difficile au sens fort par restriction au problème de la coupe maximum.

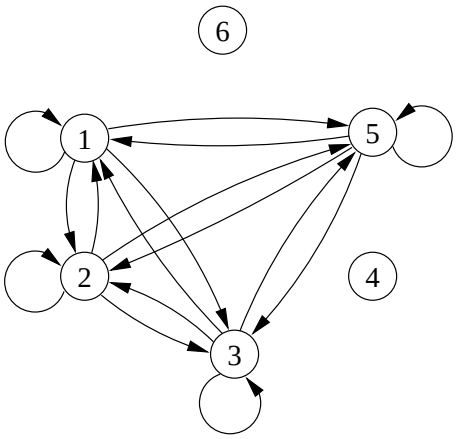


FIGURE 5.2 – Une 4-clique sur P^6_{PLO}.

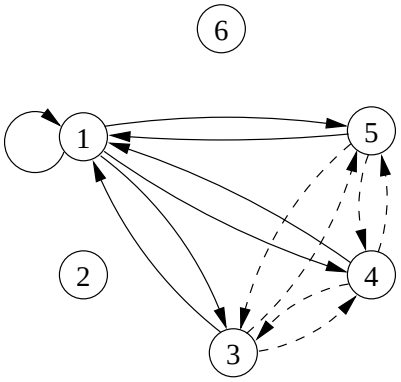


FIGURE 5.3 – Un 3-monocycle sur P^6_{PLO}. Les arcs en pointillés sont valués par -1.

Soit $i_0 \in \{1,\dots,n\}$, $j_0 \in \{1,\dots,n\} \setminus \{i_0\}$, $\emptyset \subset I \subset \{1,\dots,n\} \setminus \{i_0, j_0\}$ et $\emptyset \subset J \subset \{1,\dots,n\} \setminus \{i_0, j_0\}$ avec $I \cap J = \emptyset$, $|I| = k$ et $|J| = l$, l'inégalité

$$\delta_{i_0 i_0} + \delta_{j_0 j_0} + \delta_{i_0 j_0} + \sum_{i \in I} (\delta_{i i_0} + \delta_{i_0 i} - \delta_{i j_0}) + \sum_{j \in J} (\delta_{j j_0} + \delta_{j_0 j} - \delta_{i_0 j})$$
$$- \sum_{i \in I} \sum_{i' \in I \setminus \{i\}} \delta_{i i'} - \sum_{j \in I} \sum_{j' \in I \setminus \{j\}} \delta_{j j'} - \sum_{i \in I} \sum_{j \in J} \delta_{j i} \leq 2$$

est valide pour P^n_{PLO} et en définit une facette. Une telle inégalité, dont nous avons illustré la structure du membre gauche sur la figure 5.4, s'appelle une *k-l-bicyclette*. Pour k et l fixés, ces inégalités sont séparables en temps polynomial, car polynomiales en nombre. Là encore, le cas général est NP-difficile au sens fort par restriction au problème de la coupe maximum.

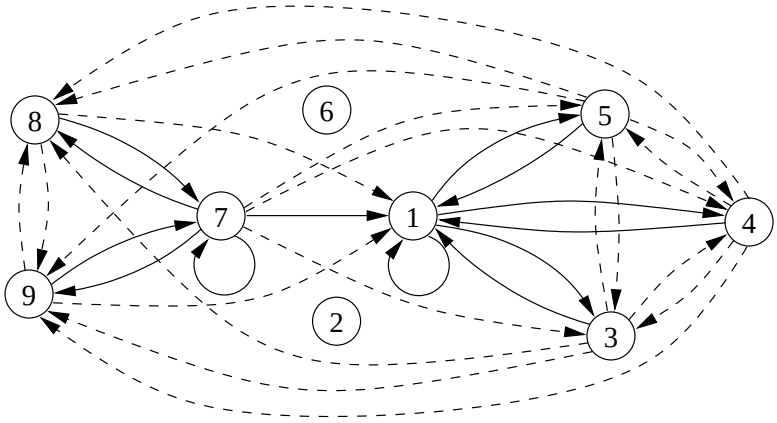


FIGURE 5.4 – Une 2-3-bicyclette sur P^6_{PLO}. Les arcs en pointillés sont valués par -1.

5.3 Sur le polytope des programmes de déplacements

5.3.1 Propriétés élémentaires

Notons tout d'abord que P^M_{PMP} est de pleine dimension sous des conditions peu restrictives : il est nécessaire et suffisant que tous les programmes qui interrompent tous les déplacements sauf deux soient admissibles. Nous faisons donc cette hypothèse dans la suite.

Par ailleurs, les inégalités (5.4) et (5.5) ainsi que les inégalités $\delta_{mm'} \geq 0$, pour tout $m, m' \in M$ avec $m \neq m'$, définissent toujours des facettes de P^M_{PMP}. Cela n'est par contre le cas ni pour les inégalités $\delta_{mm} \geq 0$, pour tout $m \in M$, et $\delta_{mm'} \leq 1$, pour tout $m, m' \in M$, ni, comme dans le cas du polytope des

sous-tournois sans circuit, pour les contraintes de transitivité (5.2). Ces dernières peuvent néanmoins être remplacées par les *contraintes de transitivité étendues*,

$$\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} + \delta_{m'm'} \leq 1 \qquad m \neq m' \neq m'' \neq m \in M, \tag{5.12}$$

qui, elles, définissent toujours des facettes de P^M_{PMP}.

5.3.2 Liens avec le polytope des sous-tournois sans circuit

L'idée était de commencer par l'étude du polytope des sous-tournois sans-circuit, d'un abord plus direct que celui des programmes de déplacements, de manière à faciliter l'étude de ce dernier.

Cette approche s'est avérée fructueuse. En effet, nous avons pu montrer que les k-cliques, les k-monocycles et les k-l-bicyclettes définissent des facettes de P^M_{PMP}, sous des hypothèses raisonnablement peu restrictives pour les k-l-bicyclettes.

Par ailleurs, soulignons que le fait que ces contraintes ne soient pas séparables en temps polynomial n'est pas un problème majeur. En effet, dans le cadre d'un algorithme de recherche arborescente polyédrique, il n'est pas nécessaire de résoudre les problèmes de séparation à l'optimum. Il arrive même parfois que, pour des histoires de performance, l'on résolve des problèmes de séparation polynomiaux de manière heuristique, c'est par exemple ce qu'à fait Kerivin (2000) pour séparer certaines contraintes de partition.

5.3.3 Contraintes de *s*- et de *t*-recouvrement

Soit $m_0 \in M$ et soient $\emptyset \subset A \subseteq T(s_{m_0})$ et $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ tels que

$$\sum_{m \in A} w_m > K_{s_{m_0}} + \sum_{m \in \bar{B}} w_m, \tag{5.13}$$

avec $\bar{B} = S(s_{m_0}) \setminus (B \cup \{m_0\})$. L'inégalité de s-recouvrement engendrée par m_0, A et B est alors définie par

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}). \tag{5.14}$$

De la même manière, soit $m_0 \in M$ et soient $A \subseteq T(t_{m_0}) \setminus \{m_0\}$ et $\emptyset \subset B \subseteq S(t_{m_0})$ tels que

$$w_{m_0} + \sum_{m \in A} w_m > K_{t_{m_0}} + \sum_{m \in \bar{B}} w_m, \tag{5.15}$$

avec $\bar{B} = S(t_{m_0}) \setminus B$. L'inégalité de t-recouvrement engendrée par m_0, A et B est alors définie par

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}). \tag{5.16}$$

Ces deux familles de contraintes sont assez naturelles : la condition (5.13) (respectivement (5.15)) exprime le fait que *tous* les déplacements de A (respectivement $A \cup \{m_0\}$) ne peuvent pas avoir été réalisés si *aucun* des déplacements de $B \cup \{m_0\}$ (respectivement B) ne l'a été ou, autrement dit, que réaliser *tous* les déplacements de $\bar{B}$ ne libère pas suffisamment de ressources pour réaliser *tous* les déplacements de A (respectivement $A \cup \{m_0\}$). Les inégalités (5.14) (respectivement (5.16)) permettent d'éviter cela, dès lors que m_0 n'est pas interrompu.

Aussi, nous avons montré que les contraintes de s- et de t-recouvrement définissent des facettes de P^M_{PMP} sous des conditions raisonnablement peu restrictives et qu'elles sont séparables en temps pseudopolynomial.

5.4 Passage à la pratique

5.4.1 Recherche arborescente polyédrique

Le passage à la pratique s'est fait par le biais d'un algorithme de recherche arborescente polyédrique (*branch-and-cut*) dont le principal ingrédient est une relaxation linéaire comprenant les $O(|M|^2)$ minicliques (5.4), les $O(|M|^2)$ mini-monocycles (5.5), les $O(|M|^3)$ contraintes de transitivité étendues (5.12) ainsi que les contraintes de s- et de t-recouvrement.

Cette relaxation, dont on peut théoriquement venir à bout en temps pseudopolynomial à l'aide de l'algorithme de l'ellipsoïde, est résolue à chaque nœud de l'arbre de recherche à l'aide d'un algorithme de coupe. Les minicliques, les mini-monocycles et les contraintes de transitivité étendues sont séparées par énumération. Les contraintes de s- et de t-recouvrement, quant à elles, sont séparées par résolution de $2|M|$ sacs à dos à l'aide de l'algorithme bien connu de Bellman (Kellerer et al., 2004).

5.4.2 Résultats expérimentaux

Nous avons illustré la pertinence pratique de cet algorithme en l'utilisant pour attaquer des instances aléatoires *difficiles*[1]. En particulier, nous avons pu résoudre exactement des instances ayant jusqu'à 119 déplacements (pour un système à 70 processeurs) en moins de quatre heures. En pratique cependant, si le temps de calcul est limité à quatre heures, l'algorithme n'a de bonnes chances de succès que jusqu'à de l'ordre de 80 déplacements.

Lorsque la taille des instances augmente, l'algorithme a néanmoins été en mesure de fournir des solutions approchées de bonne qualité, typiquement *prouvées* ne se situer qu'à moins de 5% (généralement moins) de l'optimum, à la quasi-totalité des instances (de taille allant jusqu'à 180 déplacements) sur lesquelles

1. À la fois en termes de taille mais aussi de capacité résiduelle des processeurs.

nous l'avons essayé, toujours sous la contrainte d'une limitation du temps de calcul à quatre heures.

Notons aussi que nous utilisons l'algorithme de recuit simulé présenté au chapitre 6 pour obtenir une bonne solution initiale, ce qui permet un premier élagage significatif de l'arborescence de recherche. Ceci illustre la complémentarité pratique entre une méthode de résolution approchée soignée et un algorithme polyédrique, complémentarité qui s'avère d'autant plus pertinente que les instances sont de grande taille.

Chapitre 6

Un algorithme de recuit

Introduction

Les travaux résumés dans ce chapitre font l'objet d'un article paru en 2009 dans le *Journal of Heuristics* (volume 15, pages 1 à 17).

Après quelques rappels sur les grands principes de la méthode du recuit simulé nous présentons un schéma général de fixation de ses paramètres basé sur la théorie de l'approximation différentielle. Nous illustrons ensuite l'application de ces résultats au problème de reconfiguation, expérimentations à l'appui.

6.1 La méthode du recuit simulé

La méthode du recuit simulé a été proposée dans les années 80 indépendamment par Kirkpatrick et al. (1983) et Cerny (1985) et, à l'origine, justifiée par analogie avec la technique dite du recuit utilisée par les physiciens afin de conduire certains systèmes physiques dans un état de basse énergie. Il s'agit d'une *méta-heuristique* (Dréo et al., 2003), autrement dit d'un principe général de construction d'algorithmes de résolution approchée pour les problèmes difficiles d'optimisation, continue comme combinatoire. Ses avantages sont d'être relativement bien comprise sur le plan théorique et de conduire à des algorithmes assez simples. Elle a par ailleurs été utilisée avec succès dans le cadre d'applications industrielles, et ce à de nombreuses reprises.

6.1.1 Énoncé de l'algorithme

Soit un problème d'optimisation combinatoire qui consiste, étant donné un ensemble de solutions $\Omega = \{\omega_1, \ldots, \omega_N\}$ ainsi qu'une *fonction économique* $c : \Omega \longrightarrow \{e_1, \ldots, e_P\}$, à trouver un élément $\omega^\star \in \Omega$ tel que $e_1 = c(\omega^\star) \leq c(\omega) \leq e_P$, pour tout $\omega \in \Omega$. De manière usuelle, $c(\omega)$ s'appelle la *valeur* de la solution ω. Aussi, on suppose donnée une *fonction de voisinage* $V : \Omega \longrightarrow 2^\Omega$.

La méthode est alors très simple à énoncer. Au départ, on commence avec une solution arbitraire. Soit ω la solution courante. À chaque itération, une solution candidate ω' est choisie uniformément dans le voisinage de la solution courante et acceptée avec une probabilité égale à $\min\left(1, e^{-\frac{c(\omega')-c(\omega)}{T}}\right)$. Le paramètre T s'appelle la *température* et tend vers 0, de manière monotone, selon une fonction appelée *loi de décroissance de la température*. La meilleure solution rencontrée durant l'exécution de l'algorithme est mémorisée et écrite lorsque la condition d'arrêt choisie est vérifiée.

6.1.2 Convergence

Bien que ce schéma converge en probabilité vers une solution optimale sous des conditions assez peu restrictives (symétrie de la fonction de voisinage, soit $\omega_j \in V(\omega_i) \Leftrightarrow \omega_i \in V(\omega_j)$, et forte connexité du graphe orienté induit), ce n'est, comme l'a démontré Hajek (1988), qu'au prix d'une décroissance de la température extrêmement (comprendre prohibitivement) lente. Qui plus est, on sait qu'il existe des instances du problème du couplage de cardinalité maximum (problème polynomial bien connu) et du problème de 3-coloriage d'un graphe dont la résolution nécessite un nombre exponentiel d'itérations (Sasaki & Hajek, 1988 ; Nolte & Schrader, 1996).

Ces résultats négatifs ne doivent pas obscurcir le tableau outre mesure : la pertinence pratique de la méthode reste remarquable ! À ce titre, on pourra consulter les articles de Johnson et al. (1989, 1991).

De plus, des résultats intéressants (bien que difficilement exploitables en pratique) ont été obtenus par Gelfand & Mitter (1985) quant à la probabilité qu'a l'algorithme du recuit simulé de visiter l'ensemble $\{\omega \in \Omega : c(\omega) \leq e\}$ (avec $e_1 \leq e \leq e_P$) au moins une fois après un nombre donné d'itérations. Enfin, Jerrum & Sorkin (1993) ont montré que l'algorithme, exécuté à une température constante bien choisie, a une très forte probabilité de trouver une bisection optimale [1] en $O(n^{2+\varepsilon})$ sur une certaine classe de graphes aléatoires. Le tableau n'est donc pas totalement noir sur le plan théorique !

6.1.3 Modélisation markovienne

Généralement, la mise en œuvre d'un algorithme de recuit simulé passe par l'étude de la chaîne de Markov qui modélise l'algorithme à température constante, alors appelé *algorithme de Metropolis* (Metropolis et al., 1953), et dont la *matrice des probabilités de passage* est donnée par (Lundy & Mees, 1986 ; Aarts & van

1. Problème NP-difficile.

Laarhoven, 1985)

$$A_{ij}(T) = \begin{cases} 0 & \text{si } \omega_j \notin V(\omega_i), \\ \frac{1}{|V(\omega_i)|} & \text{si } \omega_j \in V(\omega_i) \text{ et } c(\omega_j) \leq c(\omega_i), \\ \frac{e^{\frac{c(\omega_i)-c(\omega_j)}{T}}}{|V(\omega_i)|} & \text{si } \omega_j \in V(\omega_i) \text{ et } c(\omega_j) > c(\omega_i), \\ 1 - \sum_{j:\omega_j \in V(\omega_i)} A_{ij}(T) & \text{si } i = j. \end{cases}$$

Sous les hypothèses de *régularité* (c'est-à-dire, dans le présent contexte, de forte connexité du graphe orienté induit par la fonction de voisinage) et d'*apériodicité*, cette chaîne admet une unique *loi stationnaire* exprimée comme suit

$$\pi_i^{(\infty)}(T) = \frac{e^{-\frac{c(\omega_i)}{T}}}{\sum_{j=1}^{N} e^{-\frac{c(\omega_j)}{T}}}. \tag{6.1}$$

Aussi,

$$\lim_{T\to 0} \pi_i^{(\infty)}(T) = \lim_{T\to 0} \frac{1}{\sum_{j=1}^{N} e^{\frac{c(\omega_i)-c(\omega_j)}{T}}} = \begin{cases} 0 & \text{si } c(\omega_i) > e_1, \\ \frac{1}{|\Omega^\star|} & \text{sinon,} \end{cases}$$

où $\Omega^\star = \{\omega \in \Omega : c(\omega) = e_1\}$.

6.2 Fixation des paramètres

6.2.1 Notion de solution (α, β)-acceptable

Nous avons introduit la notion de solution (α, β)-acceptable, où $\beta \in [0, 1]$ définit une exigence de qualité et où $\alpha \in [0, 1]$ est la probabilité de satisfaire cette exigence. Formellement, une solution ω d'un problème de minimisation est (α, β)-acceptable si

$$P(c(\omega) \leq e_1 + \beta(e_P - e_1)) \geq \alpha.$$

Le paramètre $1-\beta$ s'interprète alors comme un *rapport d'approximation différentiel* (Demange & Paschos, 1996 ; Monnot et al., 2003). L'idée, en différentiel, consiste à ne pas seulement approcher la meilleure solution, mais aussi à s'éloigner de la pire. Le rapport différentiel situe alors la valeur d'une solution sur l'intervalle $[e_1, e_P]$ tel qu'illustré sur la figure 6.1.

De la modélisation markovienne de l'algorithme de Metropolis découle alors le résultat suivant : soit $e_1 \leq z \leq e_P$, les solutions issues de la loi stationnaire de l'algorithme de Metropolis à la température

$$T_f(z) = \frac{\beta(e_P - z)}{\log N - \log(1 - \alpha)} \tag{6.2}$$

sont (α, β)-acceptables. Il en résulte que la séquence décroissante des valeurs de la meilleure solution rencontrée par l'algorithme, $e_1 \leq z^\star \leq e_P$, engendre une séquence croissante de températures d'arrêt (cf. figure 6.3), $T_f(z^\star)$.

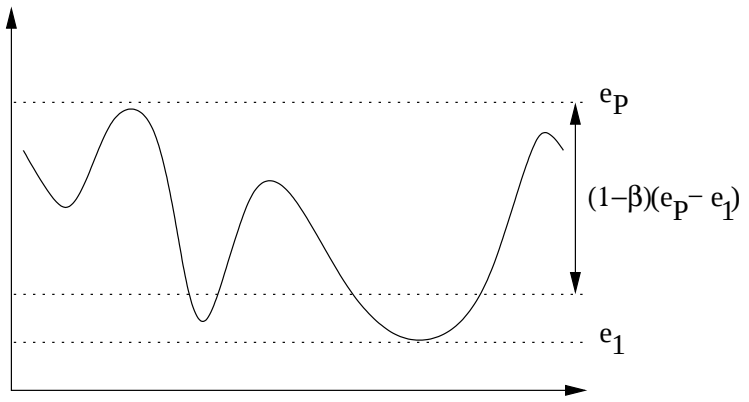


FIGURE 6.1 – Principe de l'approximation différentielle.

6.2.2 Loi de décroissance

Afin d'obtenir des solutions (α, β)-acceptables, il convient de chercher à simuler avec une précision acceptable la loi stationnaire à la température $T_f(z^\star)$. Pour ce faire, l'idée consiste à démarrer l'algorithme à une température relativement élevée, température à laquelle la convergence vers la loi stationnaire est extrêmement rapide[2], et à faire décroître la température de telle manière que les lois stationnaires associées à deux valeurs successives soient proches. Typiquement, en suivant Aarts & van Laarhoven (1985), on choisira

$$T_{k+1} = \frac{T_k}{1 + \frac{\log(1+\delta)}{e_P+1} T_k}, \tag{6.3}$$

ce qui garantit que

$$|\pi_i^{(\infty)}(T_k) - \pi_i^{(\infty)}(T_{k+1})| \leq \delta,$$

où δ est un petit réel positif. En conséquence, il est raisonnable d'escompter qu'après avoir fait décroître la température, il suffit d'un petit nombre d'itérations à température constante (on parle de *palier de température*) pour se rapprocher de la nouvelle loi stationnaire, de manière satisfaisante. Bien entendu, ce raisonnement est de nature heuristique et il existe d'autres façons de procéder. Un état de l'art a récemment été dressé par Triki et al. (2005).

6.2.3 Complexité des algorithmes résultants

Si l'on fixe α, β et δ alors on peut montrer que l'utilisation de la loi de décroissance (6.3) conjointement à la température d'arrêt $T_f(z^\star)$ induit un nombre

2. Rappelons que la convergence vers la loi stationnaire est géométrique et que la vitesse de convergence dépend de la valeur absolue de la deuxième plus grande valeur propre de la matrice des probabilités de passage (Kemeny & Snell, 1960 ; Ross, 1996). Malheureusement, celle-ci est extrêmement faible à basse température.

de paliers inférieur ou égal à

$$\frac{(1+e_P)\left(\log N - \log(1-\alpha)\right)}{\beta \log(1+\delta)} - \frac{1}{\gamma e_P}, \tag{6.4}$$

où $\gamma = \frac{\log(1+\delta)}{1+e_P}$.

Si $\log N$ est en $O(n^k)$, où n dénote la taille naturelle du problème, alors le nombre de paliers est en $O(e_P n^k)$, c'est-à-dire polynomial ou pseudopolynomial en n selon si e_P est en $O(n^l)$ ou pas. Dès lors que le nombre d'itérations par palier et que le calcul de la fonction économique sont polynomiaux en n, on obtient un algorithme polynomial ou pseudopolynomial.

Nous avons donc obtenu un schéma assez naturel de fixation des paramètres de l'algorithme du recuit simulé [3] qui conduit à des algorithmes efficaces.

6.3 Application au problème de reconfiguration

6.3.1 Aperçu de l'algorithme

Nous avons appliqué les considérations théoriques des sections précédentes au problème de reconfiguration. Dans notre cas, Ω est l'ensemble des $|M|!$ permutations des déplacements de M et deux telles permutations sont voisines si l'une peut s'obtenir à partir de l'autre en échangeant les positions de deux déplacements, et réciproquement. La forte connexité du graphe orienté induit par cette relation de voisinage est évidente.

On a $e_P = \sum_{m\in M} c_m$ (la pire solution consiste à interrompre tous les déplacements), d'où , d'après l'équation (6.2),

$$T_f(c^\star) = \frac{\beta\left(\sum_{m\in M} c_m - c^\star\right)}{\log|M|! - \log(1-\alpha)}.$$

Enfin, le calcul de la valeur associée à une permutation des déplacements est en $O(|M|)$ (il y a tout de même quelques subtilités, détaillées dans Sirdey et al., 2009, afin d'éviter de systématiquement manquer l'optimum sur certaines instances) et l'on réalise $|M|$ itérations par palier.

Dans la mesure où, d'après l'équation (6.4) et le fait que $\log n! \approx n \log n$, le nombre de paliers est en $O\left(|M| \log|M| \sum_{m\in M} c_m\right)$, l'algorithme est en

$$O\left(|M|^3 \log|M| \sum_{m\in M} c_m\right),$$

donc pseudopolynomial.

3. Méthode souvent critiquée pour son trop grand nombre de paramètres, y compris, quelque peu naïvement, dans Sirdey et al. (2003).

La figure 6.2 illustre la convergence de l'algorithme ($\alpha = 0.95$, $\beta = 0.05$ et $\delta = 0.1$) sur une instance à 14 processeurs et 56 déplacements avec $\sum_m c_m = 1227$. Il existe une solution de valeur nulle et le seuil de 5%-acceptabilité (trait horizontal sur la figure 6.2) est de 61.35. La meilleure solution rencontrée par l'algorithme a pour valeur 33 et se situe à 2.69% de l'optimum. Le temps de calcul était de 39.096 s sur un PC portable des plus moyen.

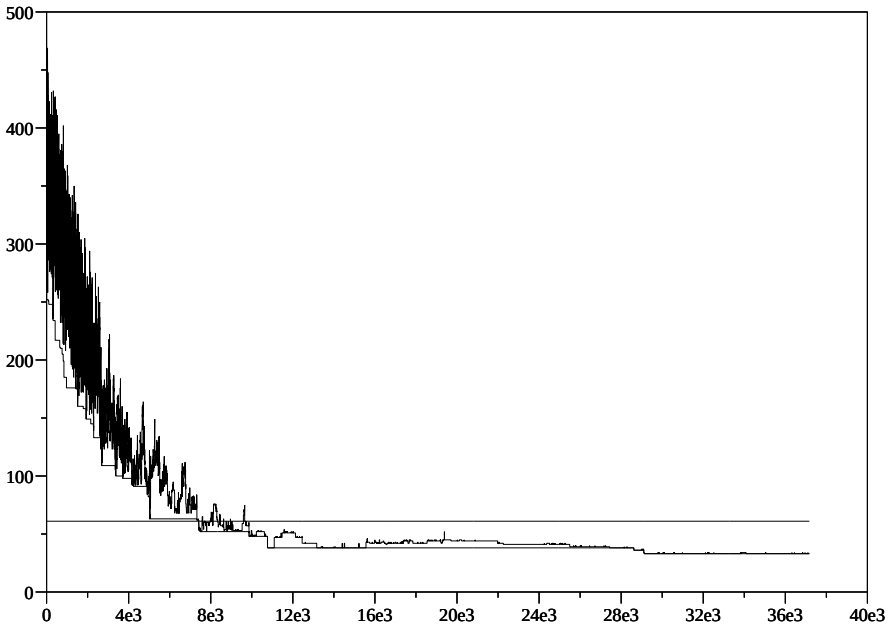


FIGURE 6.2 – Illustration de la convergence de notre algorithme de recuit simulé.

6.3.2 Résultats expérimentaux

Lors de nos expérimentations, nous avons fixé les paramètres de l'algorithme comme suit : $\alpha = 0.95$, $\beta = 0.05$ et $\delta = 0.1$. L'évaluation de l'algorithme a été réalisée sur un ensemble de 1020 instances *difficiles* (de 2 à 14 processeurs et de 10 à 254 déplacements). La table 6.1 résume les conclusions que nous avons pu tirer à l'aide des résultats (solutions optimales ou bornes inférieures) obtenus grâce à nos algorithmes de résolution exacte, en termes de distance à l'optimum. Par exemple, une solution 5%-acceptable a été obtenue pour 97.74% des instances de la base, soit 997 instances.

Pour 20 des 23 instances pour lesquelles une solution 5%-acceptable n'a pas été obtenue, il a suffi de relancer l'algorithme 1.7 fois, en moyenne, afin d'obtenir

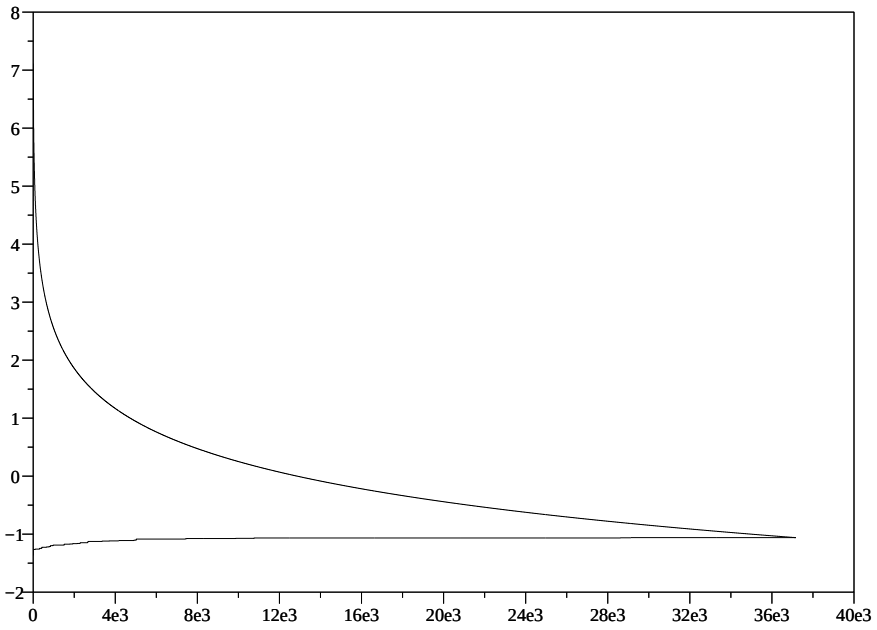


FIGURE 6.3 – Décroissance de la température (courbe du haut) et croissance de la température finale (courbe du bas), au gré des paliers (et de l'amélioration de la fonction économique). L'axe des ordonnées est logarithmique.

β	$\leq 5\%$	$]5\%, 6\%]$	$]6\%, 7\%]$
%	97.74	1.76	0.49

TABLE 6.1 – Performances de l'algorithme de recuit simulé, en termes de distance à l'optimum.

une telle solution. Au final, seules trois instances sont restées ouvertes, *instances pour lesquelles nous ne connaissons pas la valeur d'une solution optimale*. La table 6.2 indique les caractéristiques de ces trois instances ainsi qu'une borne supérieure sur la distance à l'optimum. Dans les trois cas, l'estimation est proche de l'objectif de 5%.

$\|U\|$	$\|M\|$	OPT	best β
13	36	?	5.65%
12	37	?	5.32%
10	24	?	5.90%

TABLE 6.2 – Caractéristiques des 3 instances restées ouvertes.

Au final, on estime que la probabilité d'obtention d'une solution 5%-acceptable, fixée à 0.95, est comprise entre 0.9408 et 0.9676 selon que l'on considère ou pas les trois instances restées ouvertes.

Troisième partie

Rapports de recherche

Introduction

Cette dernière partie est organisée sous la forme d'un recueil de rapports de recherche dont la majorité a donnée lieu à des articles dans des journaux spécialisés. L'ensemble de ces rapports représente notre contribution.

Page 103, nous avons reproduit le rapport « *On a resource-constrained scheduling problem with application to distributed systems reconfiguration* ». Ce rapport a donnée lieu à un article paru en 2007 dans le *European Journal of Operational Research* (volume 183, pages 546 à 563). Ce rapport est fondateur au sens où c'est le premier que nous avons écrit sur le problème de reconfiguration. En conséquence, il contient une formalisation précise du problème (qui sera reprise dans la majeure partie de nos autres rapports), une étude bibliographique détaillée ainsi que le résultat de complexité évoqué à la section 3.3. En sus, le rapport présente les méthodes de résolution exacte dont nous avons résumé les principales caractéristiques au chapitre 4.

Page 129, nous avons reproduit l'article « *Approximate resolution of a resource-constrained scheduling problem* ». Ce rapport a donné lieu à un article paru en 2009 dans le *Journal of Heuristics* (volume 15, pages 1 à 17). Ce rapport présente nos travaux sur la résolution approchée du problème le reconfiguration à l'aide de la métaheuristique du recuit simulé. Ces travaux sont résumés au chapitre 6.

Aux pages 149, 171 et 193 sont respectivement reproduits les rapports « *Polyhedral combinatorics of a resource-constrained ordering problem part I—On the partial linear ordering polytope* », « *Polyhedral combinatorics of a resource-constrained ordering problem part II—On the process move progam polytope* » et « *A branch-and-cut algorithm for a resource-constrained scheduling problem* ». Ce dernier rapport a donné lieu à un article paru en 2007 dans le journal *RAIRO Operations Research* (volume 41, pages 235 à 251). Ces trois rapports forment le corps de notre contribution sur le plan polyèdres. Cette contribution est résumé au chapitre 5.

Stricto sensu, le rapport « *Combinatorial optimization problems in wireless switch design* », qui a donné lieu à un article paru en 2007 dans le journal *4OR* (volume 5, pages 319 à 333), que nous avons reproduit page 211, ne fait pas partie de notre travail de thèse. Il nous a néanmoins semblé opportun de l'inclure dans ce mémoire dans la mesure où il illustre bien le type de problèmes combinatoires que nous, équipe d'architecture BSC, avons régulièrement à traiter.

À ces publications s'ajoute un article, « *A GRASP for a resource-constrained scheduling problem* », paru en 2010 dans l'*International Journal of Innovative Computing and Applications* (volume 2, pages 143 à 149).

Aussi, nous avons publié quelques communications que nous avons choisi de ne pas reproduire dans ce mémoire, en particulier au septième congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (Roadéf) qui s'est tenu à Lille les 6, 7, 8 Février 2006 et aux troisièmes Journées Polyèdres et Optimisation Combinatoire qui se sont tenues à Avignon les 1er et 2 Juin 2006. En sus, un résumé de notre travail de thèse est paru en 2008 dans le journal *4OR* (volume 6, pages 195 à 198).

Enfin, nous avons publié quelques articles à vocation pédagogique que nous avons aussi choisi de ne pas reproduire dans ce mémoire. L'article « Sudokus et algorithmes de recuit », paru dans le numéro d'octobre-décembre 2006 de *Quadrature* (le magazine de mathématiques pures et épicées), illustre l'art de concevoir des algorithmes de recuit sur l'application ludique d'actualité qu'est la résolution de grilles de Sudoku et l'article « Sudokus et programmation linéaire », paru dans le numéro de janvier-mars du même magazine, illustre l'omniprésence de la programmation linéaire en optimisation combinatoire à travers la présentation d'une heuristique de résolution de ce problème. Ces deux articles ont d'ores et déjà servi de source d'inspiration pour un TP d'optimisation discrète à l'Istil [4, 5]. En sus, un article de vulgarisation sur l'approche polyédrale, « Optimiser... En découpant des polyèdres », est paru dans le numéro de juillet 2007 de *L'Ouvert* (le journal de l'Apmep [6] d'Alsace et de l'Irem [7] de Strasbourg). Nous avons par ailleurs été invité à rédiger une version courte de ce dernier article, « Des solutions pour faire bonne figure », pour la rubrique W^{xyz} du numéro d'avril 2007 du magazine *La Recherche*.

4. Institut des Sciences et Techniques de l'Ingénieur de Lyon.

5. C'est à Madame Sophie Constans, chercheur à l'Institut National de Recherche sur les Transports et leur Sécurité, que nous devons cet honneur.

6. Association des Professeurs de Mathématiques de l'Enseignement Public.

7. Institut de Recherche sur l'Enseignement des Mathématiques.

On a resource-constrained scheduling problem with application to distributed systems reconfiguration * †

Renaud Sirdey[a] [b], Jacques Carlier[b], Hervé Kerivin[c] and Dritan Nace[b]

[a] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.
[b] UMR CNRS Heudiasyc (Université de Technologie de Compiègne), Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France.
[c] UMR CNRS Limos (Université de Clermont-Ferrand II), Complexe scientifique des Cézeaux, 63177 Aubière Cedex, France.

Submitted on October 18, 2005
Revised on July 14, 2006
Accepted on October 18, 2006

Abstract

This paper is devoted to the study of a resource-constrained scheduling problem, the *Process Move Programming problem*, which arises in relation to the operability of certain high availability real-time distributed systems. Informally, this problem consists, starting from an arbitrary initial distribution of processes on the processors of a distributed system, in finding the least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint is that the capacity of the processors must not be exceeded during the reconfiguration. After a brief survey of the literature, we prove the NP-hardness of the problem and exhibit a few polynomial special cases. We then present a branch-and-bound algorithm for the general case along with computational results demonstrating its practical relevance. The paper is concluded by a discussion on further research.

Keywords: combinatorial optimization, scheduling, branch and bound, distributed systems, OR in telecommunications.

∗. Nortel GSM Access R&D Technical Report PE/BSC/INF/015593 V01/EN, appeared in the European Journal of Operational Research 183:546-563, 2007.

†. This research was supported in part by Association Nationale de la Recherche Technique grant CIFRE-121/2004. Part of this work was done while the third author was working at the Institute for Mathematics and its Applications (IMA), University of Minnesota, Minneapolis, USA.

1 Introduction

Let us consider a distributed system composed of a set U of *processors* and let R denote the set of *resources* they offer. For each processor $u \in U$ and each resource $r \in R$, $C_{u,r} \in \mathbb{N}$ denotes the amount of resource r offered by processor u. We are also given a set P of applications, hereafter referred to as *processes*, which consume the resources offered by the processors. The set P is sometimes referred to as the *payload* of the system. For each process $p \in P$ and each resource $r \in R$, $w_{p,r} \in \mathbb{N}$ denotes the amount of resource r which is consumed by process p. Note that neither $C_{u,r}$ nor $w_{p,r}$ vary with time. Also, when $|R| = 1$, $C_{u,r}$ and $w_{p,r}$ are respectively denoted C_u and w_p (this principle is applied to other quantities throughout this paper).

An *admissible state* for the system is defined as a mapping $f : P \longrightarrow U \cup \{u_\infty\}$, where u_∞ is a dummy processor having infinite capacity, such that for all $u \in U$ and all $r \in R$ we have

$$\sum_{p \in P(u;f)} w_{p,r} \leq C_{u,r}, \tag{1}$$

where $P(u; f) = \{p \in P : f(p) = u\}$. The processes in $\bar{P}(f) = P(u_\infty; f)$ are not instantiated, when this set is non empty the system is in *degraded mode*.

An instance of the *Process Move Programming* (PMP) problem is then specified by two arbitrary system states f_i and f_t and, roughly speaking, consists in, starting from state f_i, finding the least disruptive sequence of operations at the end of which the system is in state f_t. The two aforementioned system states are respectively referred to as the *initial system state* and the *final system state* or, for short, the *initial state* and the *final state*[1].

Figure 1 provides an example of an instance of the PMP problem for a system with 10 processors, one resource and 46 processes. The capacity of each of the processors is equal to 30 and the sum of the consumptions of the processes is 281. The top and bottom figures respectively represent the initial and the final system states. For example, process number 23 must be moved from processor 2 to processor 6.

A process may be moved from one processor to another in two different ways: either it is *migrated*, in which case it consumes resources on both processors for the duration of the migration and this operation has virtually no impact on service, or it is *interrupted*, that is removed from the first processor and later restarted on the other one. Of course, this latter operation has an impact on service. Additionally, it is required that the capacity constraints (1) are always satisfied during the reconfiguration and that a process is moved (i.e., migrated

1. Throughout the rest of this paper, it is assumed that $\bar{P}(f_i) = \bar{P}(f_t) = \emptyset$. When this is not true the processes in $\bar{P}(f_t) \setminus \bar{P}(f_i)$ should be stopped before the reconfiguration, hence some resources are freed, the processes in $\bar{P}(f_i) \setminus \bar{P}(f_t)$ should be started after the reconfiguration and the processes in $\bar{P}(f_i) \cap \bar{P}(f_t)$ are irrelevant.

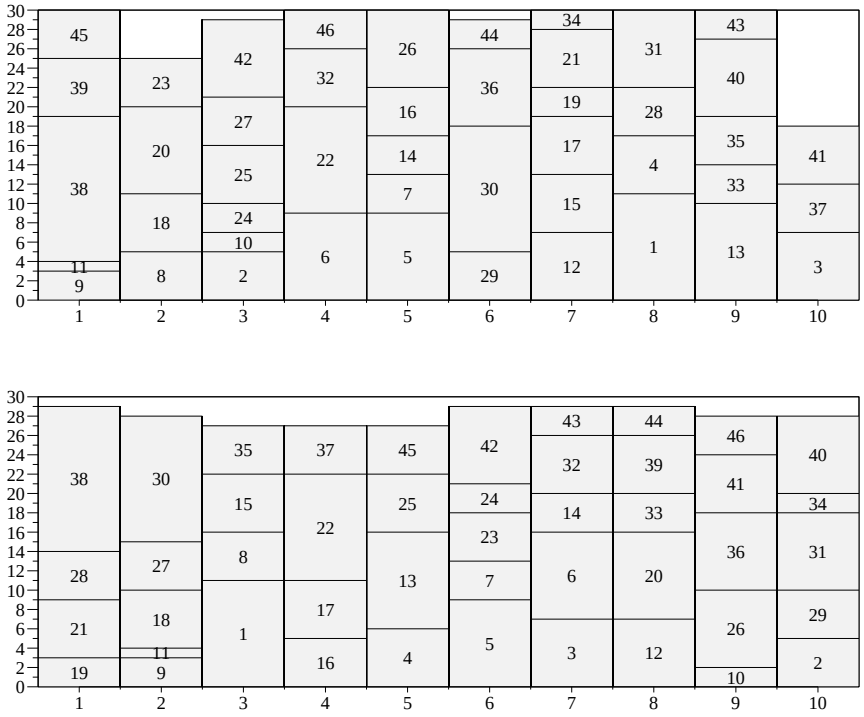


Figure 1: Example of an instance of the PMP problem.

or interrupted) at most once. The latter constraint is motivated by the fact that a process migration is far from being a lightweight operation (for reasons related to distributed data consistency which are out of the scope of this paper), as a consequence, it is desirable to avoid processes hopping around processors.

Throughout this paper, when it is said that a move is *interrupted*, it is meant that the process associated to the move is interrupted. This slightly abusive terminology significantly lightens our discourse. Additionally, it is now assumed that $|R| = 1$, unless otherwise stated.

For each processor u, a process p in $P(u; f_i) \setminus P(u; f_t)$ must be moved from u to $f_t(p)$. Let M denote the set of process moves. Then for each $m \in M$, w_m, s_m and t_m respectively denote the amount of resource consumed by the process moved by m, the processor from which the process is moved that is the *source* of the move and the processor to which the process is moved that is the *target* of the move. Lastly, $S(u) = \{m \in M : s_m = u\}$ and $T(u) = \{m \in M : t_m = u\}$.

A pair (I, σ), where $I \subseteq M$ and where $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ is a bijection, defines an admissible *process move program*, if provided that the moves in I are interrupted (the interruptions are performed at the beginning) the other moves can be performed according to σ without inducing any violation of the capacity constraints (1). Formally, (I, σ) is an admissible program if for all $m \in M \setminus I$ we have

$$w_m \leq K_{t_m} + \sum_{\substack{m' \in I \\ s_{m'} = t_m}} w_{m'} + \sum_{\substack{m' \in S(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'}, \tag{2}$$

where $K_u = C_u - \sum_{p \in P(u; f_i)} w_p$, thereby guaranteeing that the intermediate states are admissible.

Also note that because the final state is admissible, we have, for each processor $u \in U$

$$K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u)} w_m \geq 0. \tag{3}$$

Let c_m denote the cost of interrupting m, the PMP problem then formally consists, given a set of moves, in finding a pair (I, σ) such that $c(I) = \sum_{m \in I} c_m$ is minimum.

After a brief survey of the literature, we study the complexity of the PMP problem and exhibit some polynomially solvable special cases. We then present a branch-and-bound algorithm for the general case along with computational results demonstrating its practical relevance.

2 Related work

The literature related to the present problem is quite scarce.

Coffman et al. (1983, 1985) seem to be the first to study a problem relatively close to ours which consists in scheduling, without preemption, a collection of large file transfers (between storage devices) so as to minimize the makespan of the overall transfer process. Each device is assumed to have the ability to communicate directly with the others. However, they consider only a *port constraint* on the devices, that is they impose a bound on the number of simultaneous file transfers a given device can engage in, and implicitly assume that the devices have infinite capacity.

Carlier (1984a,b) studies a problem of scheduling debt payments. Although the context obviously differs, this problem is quite close to the PMP problem. Given that each person has an initial capital as well as both debts and credentials, the *debt payment problem* asks for an admissible debt payment program, that is an ordering of the debt payments such that the capital of each person always remains positive and such that all the debts end up being paid. Carlier then shows that if a payment must be performed in one go then the problem of finding such a program or deciding that none exists is strongly NP-complete and exhibits a polynomial algorithm which solves the problem when this constraint is relaxed (i.e., when the debts are *breakable*). In fact, it is possible to interpret a debt between two persons as a process move between two processors [2] (from the source processor, associated to the creditor, to the target one, associated to the debtor) but *not* vice versa. Indeed, in Carlier's model, there can be only one debt from one person to another but not the other way around (otherwise the two debts partially cancel leaving either one or no debt at all). Furthermore, the other notions involved in the definition of the PMP problem (e.g., the interruption of a process) do not really have a counterpart in the work of Carlier. Lastly, it should be emphasized that Carlier's algorithm for the breakable debt payment problem can be used to design a polynomial algorithm which solves the homogeneous case studied in Section 4.2, in the special case where the digraph underlying the instance is asymmetric (i.e., under the constraint that when some processes must be transferred from a processor A to another processor B, no process has to be transferred from B to A). Additionally, Carlier's algorithm, which is based on network flow techniques, is in essence very different from the algorithm presented in Section 4.2, which exploits strong connectivity and eulerianity properties.

Gavish & Sheng (1990) study the problem of dynamically optimizing the performances of distributed systems, such as computerized airline reservation systems, using dynamic migrations of files or database fragments in reaction to temporary changes in usage patterns. They also stress that neither their study nor most studies anterior to theirs have taken capacity constraints into

2. It follows that the NP-completeness of the PMP problem (Section 3) can also be established by restriction to the debt payment problem. However, the NP-completeness result of Carlier (1984a,b) does not allow to establish the NP-completeness of the PMP problem for a system with only two processors, in that sense Proposition 1 is a stronger result as far as the PMP problem is concerned.

account and that an assessment of the impact of such constraints on distributed file management policies is an important open issue.

More recently, Hall et al. (2001) ; Saia (2001) ; Anderson et al. (2001) have studied various flavours of a problem, referred to as the *data migration* problem, which consists in computing an efficient plan for moving objects stored on devices in a fully connected network from one configuration to another. On top of requiring that each device is involved in the transfer of only one object at a time, they explicitly consider capacity constraints on each of the devices and assume both that the objects have the same size and that there is at least one free space on each storage device in the initial as well as in the final configuration. Lastly, they also introduce the notion of *bypass node*, which is an extra storage device that can be used to store objects temporarily, and study the influence of allowing indirect migrations (via a bypass node) on the makespan of the reconfiguration.

Aggarwal et al. (2003) introduce the *load rebalancing* problem which, given a suboptimal assignment of jobs to processors, asks to relocate a subset of the jobs so as to decrease the makespan, that is the load of the heaviest loaded processor. Among other results, they propose several efficient approximation algorithms for a variant of the problem which asks to achieve the best possible makespan under the constraint that no more than k jobs are relocated. They do not, however, have to consider capacity constraints on the processors as the system reconfiguration is performed by removing all the relocated jobs and by subsequently restarting them on the appropriate processors.

It turns out that the PMP problem is quite different from the above problems. In most of the aforementioned studies the objective is to minimize the duration of the reconfiguration under a set of constraints on the legal parallelism and, sometimes, only under quite loose capacity constraints. On the contrary, in the PMP problem we are interested only in minimizing the impact the reconfiguration has on service under multidimensional capacity constraints, although most of this paper considers the monodimensional case.

3 Complexity

In this section, we study the computational complexity of the PMP problem and show, perhaps not surprisingly, that it is NP-hard in the strong sense.

Given a set of moves, say M, we focus on the decision problem, hereafter referred to as the *Zero-Impact Process Move Programming* (ZIPMP) problem, which asks whether or not there exists a bijection $\sigma : M \longrightarrow \{1, \ldots, |M|\}$ such that for all $m \in M$

$$w_m \leq K_{t_m} + \sum_{\substack{m' \in S(t_m) \\ \sigma(m') < \sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m) \\ \sigma(m') < \sigma(m)}} w_{m'}. \tag{4}$$

Recall that the 3-partition problem is the decision problem which asks, given a set E of $3k$ items, an upper bound $W \in \mathbb{N}$ and a size $s : E \longrightarrow \mathbb{N}$ such that $\frac{W}{4} < s(e) < \frac{W}{2}$ for all $e \in E$ and such that $\sum_{e \in E} s(e) = kW$, whether or not there exists a partition of E into k disjoint sets $E_1, \ldots, E_k$ such that for all $1 \leq i \leq k$

$$\sum_{e \in E_i} s(e) = W.$$

It is well-known (Garey & Johnson, 1979) that the 3-partition problem is NP-complete in the strong sense.

Proposition 1 *The ZIPMP problem is NP-complete in the strong sense, even for a system with only two processors.*

Proof. Let us consider a system composed of two processors, A and B, such that the set of process moves from A to B, denoted M_A, contains $k-1$ moves which satisfy $w_m = W \in \mathbb{N}$ and such that the set of process moves from B to A, denoted M_B, contains $3k$ moves satisfying $\frac{W}{4} < w_m < \frac{W}{2}$ and $\sum_{m \in M_B} w_m = kW$. Additionally, $K_A = W$ and $K_B = 0$. See Figure 2.

All we need to prove is that the above instance is a yes-instance if and only if there exists a partition of M_B into k disjoint sets $M_1, \ldots, M_k$ such that for all $1 \leq i \leq k$

$$\sum_{m \in M_i} w_m = W. \tag{5}$$

First suppose that such a partition does exist. It is then easy to construct a solution by first performing all the moves in any one of the M_i (this is possible since $K_A = W$) and this frees enough room on processor B to perform any one of the moves in M_A. After performing this step $k-1$ times, all the moves in M_A have been performed, so have the moves in all but one of the M_i's and there are W free units on A. Hence, by equation (5), the moves in the last of the M_i's are possible.

Conversely, let us suppose that such a partition does not exist. Let k' denote the greatest integer such that there exists $M_1, \ldots, M_{k'}$ disjoint sets which satisfy equation (5) for all $1 \leq i \leq k'$. Necessarily $k' < k-1$ (otherwise the non-existence assumption is falsified), hence it is possible to realize k' of the $k-1$ moves in M_A. Then W free units are available on A but since there exists no more set satisfying equation (5) it is only possible to transfer less than W units from B to A, it is therefore impossible to free enough room on B to perform another of the remaining moves in M_A.

Hence, the 3-partition problem can be solved by an algorithm able to solve the ZIPMP problem. The NP-completeness of the latter problem therefore follows by restriction to the 3-partition problem, itself NP-complete in the strong sense.□

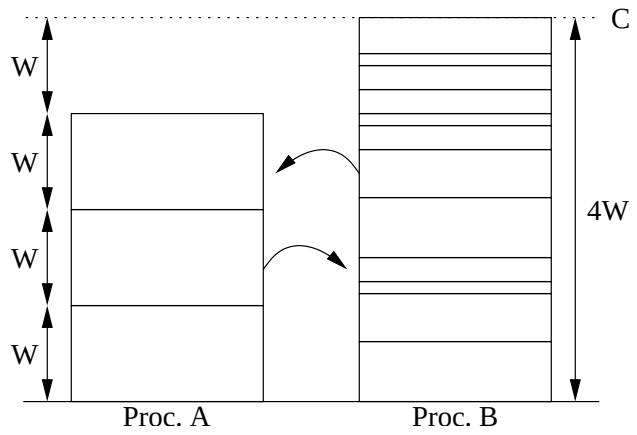


Figure 2: Illustration of the kind of instances considered in the proof of Proposition 1.

The strong NP-hardness of the PMP problem directly follows from the above proposition. As a consequence, there neither exists a polynomial nor a pseudopolynomial algorithm for the PMP problem unless $P = NP$.

Lastly, it is interesting to note that the complexity result of Carlier (1984a,b) implies that the PMP problem remains strongly NP-hard even when the digraph underlying the instance is asymmetric i.e., when there is at most one process to transfer in between each (unordered) pairs of processors.

4 Polynomially solvable special cases

This section is devoted to the study of two polynomially solvable special cases of the PMP problem.

To avoid any ambiguities we first recall a few basic notation and definitions regarding directed graphs. This terminology is borrowed from Bang-Jensen & Gutin (2002). Let $D = (V, A)$ denote a *directed multigraph* (that is parallel arcs are allowed but loops are forbidden). For a vertex $v \in V$, $N_D^+(v)$, $N_D^-(v)$, $d_D^+(v)$ and $d_D^-(v)$ respectively denote the *out-neighbourhood* (that is the set of vertices dominated by v), the *in-neighbourhood* (that is the set of vertices which dominate v), the *out-degree* (that is the number of arcs with tail v) and the *in-degree* (that is the number of arcs with head v) of v. A *walk* is an alternating sequence of vertices and arcs, say $v_1 a_1 v_2 a_2 v_3 \ldots v_{n-1} a_{n-1} v_n$, such that for $1 \leq i < n$ the tail of a_i is v_i and the head of a_i is v_{i+1}. A *closed walk* is a walk such that $v_1 = v_n$, a *trail* is a walk in which all arcs are distinct, a *path* is a trail in which all vertices are distinct and a *directed cycle* is a closed trail in which all vertices but the first and last are distinct (for short, the term *cycle* is used in the sequel).

Let M denote the set of process moves. To an instance of the PMP problem we associate a directed multigraph, denoted D and called the *transfer multigraph*,

whose vertices are associated to the processors and such that an arc (s_m, t_m) is associated to each move $m \in M$. Given a transfer multigraph we also define the *transfer digraph*, denoted $\tilde{D}$, as the directed graph obtained by contracting the parallel arcs in D.

4.1 Acyclic transfer digraphs

Our first concern is the case where the transfer multigraph is acyclic, without any constraint on the number of resources. Recall that every acyclic multigraph has a *topological ordering* of its vertices, that is there exists a bijection $\eta : V \longrightarrow \{1, \ldots, |V|\}$ such that $\eta(v) < \eta(w)$ for all arcs $(v, w) \in A$.

Proposition 2 *If D is acyclic, a zero-impact process move program exists and can be found in linear time.*

Proof. By definition of a topological ordering $\eta^{-1}(|V|)$ has no out-neighbour. Equivalently, $S(\eta^{-1}(|V|)) = \emptyset$. Hence equation (3) becomes

$$\sum_{m \in T(\eta^{-1}(|V|))} w_m \leq K_{\eta^{-1}(|V|)},$$

which means that all the moves which target $\eta^{-1}(|V|)$ are possible.

Let $1 \leq i < |V|$, then, for all j such that $i < j \leq |V|$, assume that the moves in $T(\eta^{-1}(j))$ have been performed and that the corresponding arcs have been removed from D. Since, by definition of a topological ordering, $\eta^{-1}(i)$ can dominate only vertices $\eta^{-1}(j)$ with $i < j$, there is no arc with tail $\eta^{-1}(i)$ left in D. Equivalently, there remains no move with $\eta^{-1}(i)$ as source. Therefore, by equation (3), all the moves in $T(\eta^{-1}(i))$ can be performed and the corresponding arcs can be removed from D.

The claim follows from the well-known fact that a topological ordering can be obtained in linear time (Bang-Jensen & Gutin, 2002). □

Figure 3 illustrates the resolution method. A topological ordering is $(5, 2, 7, 6, 1, 8, 3, 4)$, so the first set of moves performed (in an arbitrary order) is the set of moves which target vertex 4, then the move which targets vertex 3 is performed and so on.

When D contains some cycles, it is still possible to derive a partial ordering of the process moves by looking at the strongly connected components of D. Recall that a directed multigraph is strongly connected either if $|V| < 2$ or if it contains a path from v to w and from w to v for each pair of distinct vertices v and w and that the strongly connected components of a directed multigraph are its maximal strongly connected subdigraphs.

Indeed, the following proposition suggests that the strongly connected components of D should be considered independently and in reverse topological order.

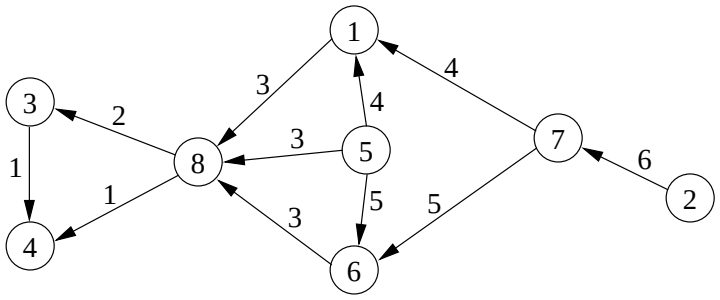


Figure 3: Illustration of the resolution method for the acyclic case.

Proposition 3 *Let $C_1, \ldots, C_n$ denote the strongly connected components of D (assumed topologically ordered). Assume that given $1 < i \leq n$ the moves having both their source and target in $\bigcup_{j=i+1}^{n} C_j$ have been performed and that the corresponding arcs have been removed from D. Then a process move program which first schedules the moves having their source in C_i and target not in C_i, then the moves internal to C_i followed by the remaining moves, dominates any other program not satisfying this property.*

Proof. Since all the moves having both their source and target in $\bigcup_{j=i+1}^{n} C_j$ have been performed the vertices targeted by the moves having their source in C_i and target not in C_i are left without any out-neighbour. Hence, these moves are possible and performing such a move frees some resources on one of the vertices of C_i therefore easing the realization of the moves internal to C_i.

So assume that the moves having their source in C_i and target not in C_i have been performed. Performing a move, say m, having its source in $\cup_{j=1}^{i-1} C_j$ and target in C_i consumes some resources on one of the vertices of C_i. Hence, doing so before performing the moves internal to C_i can only harden the realization of these moves. Additionally, m is guaranteed to become possible after the moves internal to C_i have been either performed or interrupted (since the vertices in C_i are then left without out-neighbour).

Lastly, the realization of a move internal to $\cup_{j=1}^{i-1} C_j$ can be postponed as the realization of such a move neither eases nor hardens the realization of the moves internal to C_i and reciprocally. □

Figure 4 illustrates the decomposition principle implied by the above proposition. First the moves targeting vertex 2 are performed in an arbitrary order, then the move targeting vertex 10, then the moves internal to A, then the moves targeting vertices of A with their source in B, and so on.

Corollary 1 *Let $C_1, \ldots, C_n$ denote the strongly connected components of D (assumed topologically ordered), a process move program which interrupts a move such that $s_m \in C_i$ and $t_m \in C_j$ with $i \neq j$ is dominated.*

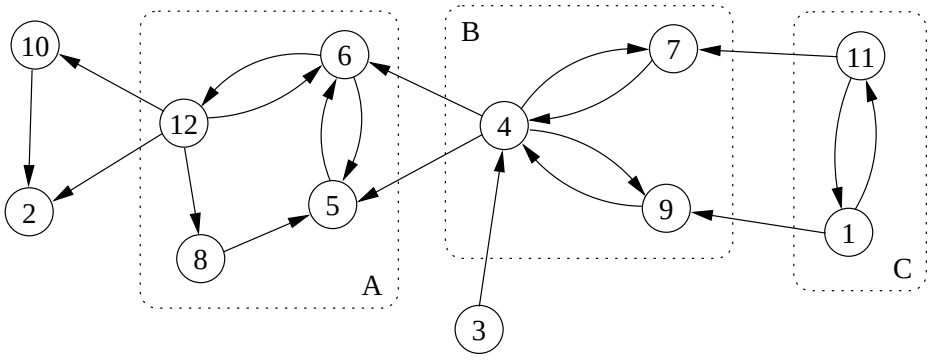


Figure 4: Illustration of the decomposition principle implied by Proposition 3.

4.2 The homogeneous case

We now turn to the case where the consumption of each of the processes is equal to a constant, supposed equal to 1 without loss of generality.

Recall that a directed multigraph is eulerian if it is connected and if $d^+(v) = d^-(v)$ for all $v \in V$ and that such a multigraph possesses an eulerian tour, that is a closed trail which uses every arc exactly once.

First we have the following proposition.

Proposition 4 *If D is eulerian then the homogeneous case can be solved in linear time.*

Proof. If there exists a processor $u \in U$ such that $K_u \geq 1$ then a zero-impact process move program is obtained by performing the moves in the reverse order of an eulerian tour on D, starting with any of the moves targeting u.

Otherwise, any one move m such that $c_m = \min_{m' \in M} c_{m'}$ is interrupted and, since this frees one unit on s_m, the remaining moves can be performed in the reverse order of an eulerian tour on D, starting with any of the moves targeting s_m and preceding m in the eulerian tour.

The claim follows from the well-known fact that an eulerian tour can be obtained in linear time (Bang-Jensen & Gutin, 2002). □

We now suppose that D is strongly connected and not eulerian and demonstrate that in this case a zero-impact process move program exists and can be found in polynomial time. We do so by studying Algorithm 1.

Lemma 1 *The moves performed at step (a) of Algorithm 1 are possible.*

Proof. The first time the loop is executed we have $C = D$ and, hence, no move satisfies the premises of step (a).

Otherwise, when D is no more strongly connected, v is left without any out-neighbour. Hence, equation (3) implies that all the moves which target v are

While $V \neq \emptyset$

Let C denote the set of vertices in the last of the (topologically ordered) strongly connected components of D.

(a) If C contain only one vertex, say v, then perform all the moves targeting v in an arbitrary order, remove them from M, remove the corresponding arcs from A and remove v from V.

(b) Else choose a vertex, say v_0, in C whose remaining capacity is non zero and a maximal eulerian subdigraph rooted at v_0, perform the moves in the subdigraph in the reverse order of an eulerian tour, removing them from M and removing the corresponding arcs from A.

End.

Algorithm 1: An algorithm for the homogeneous case when D is strongly connected and non-eulerian.

possible. □

Lemma 2 *The first time step (b) of Algorithm 1 is executed, there exists a vertex v_0 in C such that $K_{v_0} > 0$.*

Proof. The first time step (b) of the algorithm is executed we have $C = D$. Since D is not eulerian there exists v_0 such that $d^+(v_0) \neq d^-(v_0)$. So either $d^+(v_0) < d^-(v_0)$ or $d^+(v_0) > d^-(v_0)$ in which case since $d^+(v_0) + \sum_{v \neq v_0} d^+(v) = d^-(v_0) + \sum_{v \neq v_0} d^-(v)$ we have $\sum_{v \neq v_0} d^+(v) < \sum_{v \neq v_0} d^-(v)$ and, by the pigeon-hole principle, there exists a vertex, say v_0' such that $d^+(v_0') < d^-(v_0')$. By equation (3), a vertex such that $d^+(v_0) < d^-(v_0)$ is such that $K_{v_0} \geq d^-(v_0) - d^+(v_0) > 0$. □

Lemma 3 *Each time step (b) of Algorithm 1 is executed, there exists a vertex v_0 in C such that $K_{v_0} > 0$.*

Proof. A strongly connected component is said to be *terminal* if it has no out-neighbour.

The lemma is established by demonstrating that, each time the loop is executed, the terminal strongly connected components of the remaining transfer multigraph either contain one vertex or contain a vertex, say v_0, such that $K_{v_0} > 0$.

Lemma 2 proves that it is initially the case.

Assume this is true at a given iteration of the algorithm.

Then if step (a) is executed new terminal strongly connected components may appear but all of these components are such that there exists a vertex v_0 with $K_{v_0} > 0$ (regardless of their cardinality). This is so because for each of

the newly introduced components at least one move having its source and target respectively in and not in the component has been performed.

If step (b) is executed, then new terminal strongly connected components may appear but they all contain only one vertex. This is so because assuming otherwise would contradict the fact that the removed eulerian subdigraph was maximal for it would mean that at least one cycle encounters at least one vertex of the subdigraph. □

The following proposition is an immediate consequence of Lemmas 1 and 3.

Proposition 5 *If D is non-eulerian and strongly connected, Algorithm 1 outputs a zero-impact process move program.*

We are now able to solve the homogeneous case.

Corollary 2 *Assume that D is connected*[3] *then, unless D is eulerian and $K_u = 0$ for all $u \in U$, a zero-impact admissible process move program exists and can be found in polynomial time.*

Proof. If D is eulerian then we proceed as in the proof of Proposition 4. So let us assume that D is connected and not eulerian and let $C_1, \ldots, C_n$ denote its strongly connected components (topologically ordered). Algorithm 1 considers the strongly connected components of D as implied by Proposition 3. Assume that $|C_n| > 1$. If the transfer multigraph, say D'_n, associated to the moves internal to C_n is not eulerian then Proposition 5 shows how to find a zero-impact process move program. Otherwise if D'_n is eulerian then $d^+_{D'_n}(v) = d^-_{D'_n}(v)$ for all vertices of D'_n however since D is connected then at least one vertex in C_n, say v_0, is the head of an arc whose tail is not in C_n it follows that $d^-_D(v_0) > d^+_D(v_0)$ and, hence, that $K_{v_0} > 0$. This provides a vertex from which an eulerian tour can be started.

When the moves internal to C_i ($i < n$, $|C_i| > 1$) are considered then, since D is connected, at least one move with source in C_i and target not in C_i has been performed, therefore ensuring that one unit of load is free on at least one of the vertices of C_i. Let D'_i denote the transfer digraph associated to the moves internal to C_i. It follows that a zero-impact process move program is given either by an eulerian tour (if D'_i is eulerian) or by Proposition 5 otherwise.

The claim follows from the fact that Algorithm 1 is clearly polynomial. □

Figure 5 illustrates the functioning of the algorithm. Initially, a maximal eulerian subdigraph rooted at 2 is chosen (dashed arcs). This is so because $d^+(2) < d^-(2)$. The moves are then performed in the reverse order of an eulerian tour on the subdigraph. After, this initial step, the remaining graph has two

3. If this assumption is not satisfied, then the argument can be repeated for each of the connected components of D.

connected components ($\{2,3\}$ and $\{4,5,7,8\}$) which can be considered independently. The latter is considered first on the example. It has 3 strongly connected components ($\{4\}$, $\{7,8\}$ and $\{5\}$, in topological order). So $\{5\}$, the last, is considered first and the move from 8 to 5 is scheduled, which frees one unit on 8 which is chosen as the root of the small maximal eulerian subdigraph (7 could have been chosen as well because $d^+(7) < d^-(7)$). The remaining graph is acyclic so we are done.

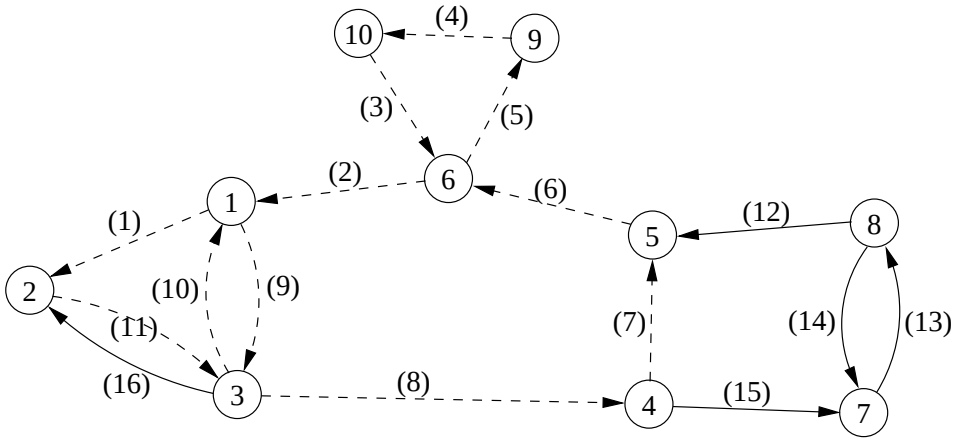


Figure 5: Illustration of the functioning of Algorithm 1.

5 A branch-and-bound algorithm

In this section, we present a branch-and-bound algorithm for the PMP problem. The algorithm initially starts with the worst possible solution, which consists in interrupting all the moves. Then an admissible program is built, each branching decision consisting in choosing an interrupted process to concatenate to the program ordering, among those for which doing so preserves the admissibility of the program. A leaf is obtained when no such process exists. This scheme is complemented by a lower bound as well as dominance relations.

We first describe each of the algorithm building blocks separately and then sketch how to integrate them in a practical branch-and-bound algorithm. Section 6 reports on computational results.

5.1 Branching scheme

A node of the search tree is denoted by as a quadruplet $N = (I, J, \sigma_J, R)$ where I, J and R respectively denote the sets of moves which are interrupted, ordered or yet neither interrupted nor ordered and where $\sigma_J : J \longrightarrow \{1, \ldots, |J|\}$ is an ordering of the moves in J.

For such a quadruplet to define an admissible node, it is required that the sets I, J and R are both mutually exclusive (that is $I \cap J = I \cap R = J \cap R = \emptyset$) and collectively exhaustive (i.e., $I \cup J \cup R = M$) as well as for $(I \cup R, \sigma_J)$ to be an admissible process move program. Stated in plain English, this latter requirement expresses the fact that as long as the moves in $I \cup R$ are interrupted, the moves in J can be performed according to σ_J without inducing any violation of the capacity constraints.

Given a node N and a processor u, let

$$\ell_u(N) = \min_{i=1,\ldots,|J|} \left(K_u + \sum_{m \in S(u) \cap (I \cup R)} w_m + \sum_{\substack{m \in S(u) \cap J \\ \sigma_J(m) \leq i}} w_m - \sum_{\substack{m \in T(u) \cap J \\ \sigma_J(m) \leq i}} w_m \right) \quad (6)$$

and

$$L_u(N) = K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u) \cap J} w_m. \quad (7)$$

Informally, $\ell_u(N)$ is the minimum remaining capacity of u during the execution of $(I \cup R, \sigma_J)$ and $L_u(N)$ is the remaining capacity of u after the execution of $(I \cup R, \sigma_J)$.

Proposition 6 *Let $N = (I, J, \sigma_J, R)$ be a node of the search tree and let $m \in R$, if $w_m \leq \ell_{s_m}(N)$ then $N' = (I, J \cup \{m\}, \sigma_{J \cup \{m\}}, R \setminus \{m\})$ is an admissible node for the search tree, where $\sigma_{J \cup \{m\}}$ is an ordering of the moves in $J \cup \{m\}$ such that $\sigma_{J \cup \{m\}}(m') = \sigma_J(m')$ for all $m' \in J$ and $\sigma_{J \cup \{m\}}(m) = |J| + 1$.*

Proof. By definition of ℓ_u, the fact that $w_m \leq \ell_{s_m}(N)$ implies that the process associated to m can remain on s_m during the entire execution of the program $(I \cup R, \sigma_J)$. After its execution, the remaining capacity on t_m is equal to

$$L_{t_m}(N) = K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} - \sum_{m' \in T(t_m) \cap J} w_{m'}$$

and, from equation (3), we have

$$K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} - \sum_{m' \in T(t_m) \cap J} w_{m'} \geq \sum_{m' \in T(t_m) \cap (I \cup R)} w_{m'} \geq w_m.$$

Hence, after all the moves in J have been performed, there is enough capacity on t_m to host the process associated to m. □

Note that the following relationships hold

$$\ell_{s_m}(N') = \ell_{s_m}(N) - w_m, \quad (8)$$

$$L_{t_m}(N') = L_{t_m}(N) - w_m, \quad (9)$$

$$\ell_{t_m}(N') = \min(\ell_{t_m}(N), L_{t_m}(N')). \quad (10)$$

Our branching scheme can then be stated as follows. The root node is $(\emptyset, \emptyset, \sigma_\emptyset, M)$ and is associated to the process moves program $(M, \sigma_\emptyset)$ which interrupts all the moves. At a node $N = (I, J, \sigma_J, R)$ of the search tree, let $I' = \{m \in R : w_m > \ell_{s_m}(N)\}$. By definition of ℓ_u, a process associated to a move m in I' cannot remain on s_m during the execution of $(I \cup R, \sigma_J)$ without inducing a violation of the capacity constraints. Hence, a move in I' cannot be added to J and concatenated to σ_J, and it will remain so in the branch rooted at N since ℓ_u is a nonincreasing function of $|J|$ (from equations (8) and (10)). It follows that for each $m \in R \backslash I'$ the nodes $N' = (I \cup I', J \cup \{m\}, \sigma_{J\cup\{m\}}, R \backslash (I' \cup \{m\}))$ are generated.

Hence, when branching from a node, the number of ordered moves is increased by one whereas the number of interrupted moves is increased by a number in $\{0, \ldots, |R| - 1\}$.

5.2 Lower bounds

At a node $N = (I, J, \sigma_J, R)$, let $\mathrm{KP}(u)$ denote the value of an optimal solution to the following knapsack problem

$$\begin{cases} \text{Maximize} \displaystyle\sum_{m \in S(u) \cap R} c_m x_m, & (11) \\ \text{s. t.} \\ \displaystyle\sum_{m \in S(u) \cap R} w_m x_m \leq \ell_u(N), & (12) \\ x_m \in \{0, 1\}, \ m \in S(u) \cap R. \end{cases}$$

We refer the reader to Kellerer et al. (2004) for details regarding the knapsack problem.

Proposition 7 *A lower bound on the values of the solutions which can be obtained by exploring the branch rooted at N is provided by*

$$LB(N) = \sum_{m \in I} c_m + \sum_{u \in U} LB(u), \tag{13}$$

where $LB(u) = W_u - KP(u)$ and $W_u = \sum_{m \in S(u) \cap R} c_m$.

Proof. Since ℓ_u is a nonincreasing function of $|J|$, the sum of the weights of the moves in $R \cap S(u)$ which can further be concatenated to σ_J cannot exceed ℓ_u. This is captured in the knapsack constraint (12). Hence, $\mathrm{KP}(u)$ provides an upper bound on the sum of the costs of the moves in $R \cap S(u)$ which can further be concatenated to σ_J. □

Fortunately, the knapsack problem is one of the easier NP-hard problems (see Pisinger, 2005 for a recent survey regarding the relative easiness of the knapsack

problem) and, in particular, it can be solved in pseudopolynomial time. For example, lower bound (13) can be obtained in $O\left(\sum_{u\in U}|S(u)\cap R|\ell_u(N)\right)$ using the well-known Bellman recursion (Bellman, 1957). Moreover, if the results of the individual knapsack problems are memorized at each depth, computing the bound at a given depth requires solving only two knapsack problems: one for the source and one for the target processor of the last move in the schedule.

When the size of the coefficients prevents the use of dynamic programming, a tight upper bound on $\mathrm{KP}(u)$ can be obtained using any FPTAS [4] for the knapsack problem leading to a slightly weaker lower bound. See for example Kellerer & Pferschy (1999).

Also, computationally cheaper, but weaker, lower bounds can be obtained from any upper bound for problem (11), the so-called Dantzig bound obtained by solving the linear relaxation of the knapsack problem would be an example. Note that when $c_m = w_m$, problem (11) becomes a subset sum problem leading to the following lower bound

$$\mathrm{LB}'(N) = \sum_{m\in I} w_m + \sum_{u\in U} \max\left(0, W_u - \ell_u(N)\right).$$

Lastly, $\mathrm{LB}(N)$ can be generalized to the multiple resource case. Problem (11) then becomes a multidimensional knapsack problem which is still reasonable to tackle using dynamic programming for a small enough number of resources (say less than or equal to 3). When the number of resources increases, however, it is likely that only upper bounds on $\mathrm{KP}(u)$ will be available. The reader is referred to Kellerer et al. (2004) for details on how to solve the multidimensional knapsack problem using dynamic programming as well as on how to obtain upper bounds.

5.3 Dominance relations

The following lemma is stated without proof.

Lemma 4 *If $a \geq c$ and $b \geq d$ then $\min(a, b) \geq \min(c, d)$.*

Proposition 8 *Let $N_1 = (I_1, J_1, \sigma_{J_1}, R_1)$ and $N_2 = (I_2, J_2, \sigma_{J_2}, R_2)$ be two nodes of the search tree, then N_1 dominates N_2 if the following conditions hold*

1. $R_1 = R_2 = R$.
2. $\sum_{m\in I_1} c_m \leq \sum_{m\in I_2} c_m$.
3. $L_u(N_1) \geq L_u(N_2)$, $\forall u \in U$.

4. Recall (Kellerer et al., 2004) that given $\varepsilon \in]0, 1[$, an ε-approximation scheme for a maximization problem is an algorithm which produces solutions of value greater than or equal to $(1-\varepsilon)\mathrm{OPT}(I)$ for all instances I of the problem. A Fully Polynomial Time Approximation Scheme (FPTAS) is an ε-approximation scheme whose running time is polynomial in the natural size of the instance as well as in $\frac{1}{\varepsilon}$.

4. $\ell_u(N_1) \geq \ell_u(N_2)$, $\forall u \in U$.

Proof. Let $N_2^\star = (I_2 \cup I^\star, J_2 \cup J^\star, \sigma_{J_2 \cup J^\star}, \emptyset)$ denote the best leaf of the branch rooted at N_2 and let $m = \sigma^{-1}_{J_2 \cup J^\star}(|J_2| + 1)$ (assuming $|J^\star| \geq 1$).

Let $N_2^{(m)} = (I_2, J_2 \cup \{m\}, \sigma_{J_2 \cup \{m\}}, R \setminus \{m\})$, since $\ell_u(N_1) \geq \ell_u(N_2)$ for all $u \in U$ the node $N_1^{(m)} = (I_1, J_1 \cup \{m\}, \sigma_{J_1 \cup \{m\}}, R \setminus \{m\})$ is admissible. Using Condition 4 and equation (8) we have

$$\ell_{s_m}(N_1^{(m)}) = \ell_{s_m}(N_1) - w_m \geq \ell_{s_m}(N_2) - w_m = \ell_{s_m}(N_2^{(m)}).$$

Using Condition 3 and equation (9) we have

$$L_{t_m}(N_1^{(m)}) = L_{t_m}(N_1) - w_m \geq L_{t_m}(N_2) - w_m = L_{t_m}(N_2^{(m)}). \tag{14}$$

Lastly, using Condition 4, equations (10) and (14) as well as Lemma 4 we have

$$\ell_{t_m}(N_1^{(m)}) = \min(\ell_{t_m}(N_1), L_{t_m}(N_1^{(m)})) \geq \min(\ell_{t_m}(N_2), L_{t_m}(N_2^{(m)})) = \ell_{t_m}(N_2^{(m)}).$$

Hence, for all $u \in U$ we have $L_u(N_1^{(m)}) \geq L_u(N_2^{(m)})$ as well as $\ell_u(N_1^{(m)}) \geq \ell_u(N_2^{(m)})$.

The above argument can be applied iteratively until the node $N_1^\star = (I_1, J_1 \cup J^\star, \sigma_{J_1 \cup J^\star}, I^\star)$ is obtained. Then the best leaf of the branch rooted at N_1 has value at most equal to

$$\sum_{m \in I_1} c_m + \sum_{m \in I^\star} c_m,$$

which is, by Condition 2, smaller than or equal to $\sum_{m \in I_2} c_m + \sum_{m \in I^\star} c_m$. □

The dominance relation of Proposition 8 generalizes several other relations.

Provided that many equivalent total orderings of a set of non-interrupted moves can be obtained by combining a given set of per-processor orderings, it is expected that a significant amount of redundancy can be removed from the search tree by considering the following special case of the dominance relation of Proposition 8. Consider two nodes $N_1 = (I, J, \sigma_J^{(1)}, R)$ and $N_2 = (I, J, \sigma_J^{(2)}, R)$. If $\sigma_J^{(1)}$ and $\sigma_J^{(2)}$ are such that, for all $u \in U$, the ordering of the moves in $J \cap (S(u) \cup T(u))$ induced by $\sigma_J^{(1)}$ is equivalent to the one induced by $\sigma_J^{(2)}$ then N_1 dominates N_2 and reciprocally. This is so because $L_u(N_1) = L_u(N_2)$ and $\ell_u(N_1) = \ell_u(N_2)$ for all $u \in U$.

The strong-connectivity-based dominance relation discussed in Section 4.1 is also taken into account by the rule of Proposition 8. For example, consider two nodes $N_1 = (I, J, \sigma_J^{(1)}, R)$ and $N_2 = (I, J, \sigma_J^{(2)}, R)$. Then for $i = 1, \ldots, |J|$ let $m = \sigma_J^{(1)-1}(i)$ and let $C_n \subseteq U$ denote the last (topologically ordered) strongly connected component of the transfer digraph induced by the moves in $\{m' \in J : \sigma(m') \geq i\}$. Assuming that $\sigma_J^{(1)}$ and $\sigma_J^{(2)}$ induce equivalent orderings of the moves in C_n, if m is always internal to C_n when $|C_n| > 1$ then we have $L_u(N_1) = L_u(N_2)$ as well as $\ell_u(N_1) \geq \ell_u(N_2)$ for all $u \in U$. Hence N_1 dominates N_2.

5.4 Subproblem selection

Subproblem selection is performed in a greedy fashion. At a node $N = (I, J, \sigma_J, R)$ of the search tree, the immediate profit associated to the decision of using a move $m \in R$ such that $w_m \leq \ell_{s_m}(N)$ for branching is defined as

$$p_m = c_m - (W_s - \mathrm{KP}_s - \mathrm{LB}(s_m)) - (W_t - \mathrm{KP}_t - \mathrm{LB}(t_m)),$$

where $W_s = \sum_{m' \in S(s_m) \cap R \setminus \{m\}} c_{m'}$, $W_t = \sum_{m' \in S(t_m) \cap R} c_{m'}$ and where KP_s and KP_t respectively denote the value of an optimal solution to knapsack problems

$$\begin{cases} \text{Maximize} \displaystyle\sum_{m' \in S(s_m) \cap R \setminus \{m\}} c_{m'} x_{m'}, \\ \text{s. t.} \\ \displaystyle\sum_{m' \in S(s_m) \cap R \setminus \{m\}} w_{m'} x_{m'} \leq \ell_{s_m}(N) - w_m, \\ x_{m'} \in \{0,1\},\ m' \in S(s_m) \cap R \setminus \{m\}, \end{cases}$$

and

$$\begin{cases} \text{Maximize} \displaystyle\sum_{m' \in S(t_m) \cap R} c_{m'} x_{m'}, \\ \text{s. t.} \\ \displaystyle\sum_{m' \in S(t_m) \cap R} w_{m'} x_{m'} \leq \min(\ell_{t_m}(N), L_{t_m}(N) - w_m), \\ x_{m'} \in \{0,1\},\ m' \in S(t_m) \cap R. \end{cases}$$

The right-hand sides of the capacity constraints of the above two problems are justified by equations (8) as well as (9) and (10), respectively.

Hence, the increment in the lower bound is taken into account when evaluating branching decisions, the moves inducing the biggest immediate profits being used for branching first.

Note that this subproblem selection scheme can be used as the basis of a simple pseudopolynomial greedy algorithm for the PMP problem.

5.5 Putting it all together

We have implemented a DFS branch-and-bound algorithm based on the ideas discussed in the previous sections, namely lower bound (13), the dominance relations of Proposition 8 as well as the subproblem selection strategy of Section 5.4.

The resolution of the knapsack problems involved in both the calculation of lower bound (13) and the subproblem selection scheme is performed using the Bellman Algorithm (Kellerer et al., 2004).

The exploitation of the dominance relation of Proposition 8 deserves more comments.

Indeed, there are three main ways of exploiting dominance relations within a branch-and-bound algorithm:

1. Exclude a node from consideration if it is dominated by a node which *has already been* considered (Ibaraki, 1977).
2. Exclude a node from consideration if there exists a node which dominates it, *regardless of whether or not* the latter has already been considered (Baptiste et al., 2004).
3. *Replace* a node by another node which dominates it, if such a node exists and can be found (Carlier & Chrétienne, 1988).

All of these strategies have pros and cons. Strategy 1 requires memorizing (at least partially) the set of nodes considered so far and may result in the exploration of redundant branches: for example if the branching procedure considers N_1 before N_2 and if N_2 dominates N_1. Strategy 2 does not require memorizing the set of nodes considered so far (as long as the dominance relation has been supplemented so as to guarantee unicity) but may result in delaying the improvement of the upper bound: for example if the branching procedure considers nodes N_1, N_2 and N_3 (in that order) and if N_3 dominates N_1 then the algorithm explores only the branches rooted at N_2 and N_3 it is however possible that exploring the branch rooted at N_1 improves the upper bound enough so that there is no need to consider N_2, so it comes down to whether it is computationally more interesting to explore the branch rooted at N_1 and the branch rooted at N_3 (despite of the fact that it is known to be redundant) or the branches respectively rooted at N_2 and N_3. Lastly, strategy 3 requires memorizing (at least partially) the set of nodes considered so far but, thanks to the fact that replacement is performed, it avoids both redundancy and delayed upper bound improvement, it however requires being able to find dominating nodes from a given node and this problem might be as hard as the problem the branching procedure is solving.

As long as the memory is managed efficiently, memorizing the set of nodes considered so far is not an issue: if the branching procedure is to succeed it must not consider too many nodes and workstations nowadays usually have fairly huge amounts of memory. Additionally, it should be emphasized that the branching procedure discussed in this paper is not destined to be embedded in a real-time system, see the discussion in Section 7.

On empirical grounds, strategy 1 appears to be the most suited to exploit the dominance relation of Proposition 8. This is performed using a balanced binary search tree (Knuth, 1998) keyed on the binary representation of the set R of a node $N = (I, J, \sigma_J, R)$, each key being associated to a list of triplets $\{c(N), L(N), \ell(N)\}$. When a node is considered, the list associated to R is searched for a triplet which dominates the node. If such a triplet is found the branch rooted at the node is pruned. Otherwise, the branch is explored. Then

the list is searched for triplets which are dominated by the triplet associated to the node, which are removed, and the latter is added at the front of the list.

6 Computational experiments

In this section, we report on computational experiments carried out so as to assess the practical relevance of the branch-and-bound algorithm of Section 5. These experiments have been performed on a Sun Ultra 10 workstation with a 440 MHz Sparc microprocessor, 512 MB of memory and the Solaris 5.8 operating system.

6.1 Instance generation

Given U the set of processors, C the processor capacity and W an upper bound on the process consumption, an instance is generated as follows.

First, the set of processes is built by drawing consumptions uniformly in $\{1, \ldots, W\}$ until $\sum_{p \in P} w_p \geq C|U|$. The initial state, f_i, is then generated by randomly assigning the processes to the processors: the processor to which a process is assigned is drawn uniformly from the set of processors whose remaining capacity is sufficient (note that not all processes necessarily end up assigned to a processor). The final state, f_t, is built in the very same way with the exception that only the processes which are assigned to a processor in the initial state are considered. An instance is considered valid only if all the processes assigned to a processor in the initial state are also assigned to a processor in the final state. Invalid instances are discarded and the construction process is repeated until a valid instance is obtained (the rejection rate depends on the parameters, as an example, coarse estimates for $|U| = 10$, $C = 100$ as well as $W = 10$ and $W = 50$ respectively are 29% and 41%). The set of moves is then built as explained in Section 1.

It should be emphasized that the above scheme generates instances for which the capacity constraints are extremely tight, instances which can be expected to be hard and, in particular, significantly harder than those occurring in practice. As an example, for $|U| = 10$, $C = 100$ and $W = 10$ only 1.28% of free capacity remains, on average, on each of the processors. However, for the system to which this work is to be applied (Sirdey et al., 2003) the maximum *theoretical* load of a processor ranges (nonlinearly) from at most 50% (for a system with 2 processors) to at most around 93% (for a system with 14 processors, which is the maximum). This is so because some spare capacity is provisioned for fault tolerance purpose and this spare capacity is spread among all the processors. Additionally, it should be stressed that the system carries at most 100 processes and that a preprocessing technique, based on the fact that the properties of a system state are invariant by a permutation of the processors, is used to decrease the number of moves

by around 25% on average. It turned out that our algorithm was able to solve virtually all practical instances within a few seconds and that, as a consequence, we had to design more aggressive instance generation schemes, such as the above, in order to push the algorithm to its limits.

Lastly, we have supposed that $c_m = w_m$, which is quite natural for our application as it is reasonable to assume that the amount of service provided by a process is proportional to the amount of resources it consumes.

6.2 Influence of the algorithm building blocks

For a small set of moderate size instances generated using the scheme of Section 6.1, Table 1 provides the number of nodes explored by the algorithm ("#nodes"), the number of entries in the binary search tree discussed in Section 5.5 ("#keys") as well as the total number of items stored in it [5] ("#items"), that is the sum over the set of entries of the length of the associated list, when only the lower bound is activated (column "LB"), when only the dominance relation is activated (column "Dom.") and when both the lower bound and the dominance relation are activated (column "LB & Dom.").

Table 1 illustrates that both the lower bound and the dominance relations significantly contribute to the reduction of the search space. It also illustrates the fact that the size of the data structure used to exploit the dominance relation grows mildly with the number of nodes.

N.	$\|M\|$	OPT	LB	Dom.			LB & dom.		
			#nodes	#nodes	#keys	#items	#nodes	#keys	#items
01	22	6	>18500000	>15900000	>316729	>606958	177542	6738	7905
02	21	17	16647308	>15500000	>224009	>454493	189618	7255	11178
03	16	23	12726	319552	8905	16232	2679	220	244
04	20	10	575391	>16100000	>210796	>510507	34829	2093	2573
05	17	26	1243750	488432	10821	22253	23635	1354	1968
06	19	25	265197	13217379	136421	480749	29891	1808	2162
07	18	5	14972721	5876920	66570	153435	55209	2685	4116
08	23	23	>23600000	>15000000	>334966	>627169	457337	18783	24298
09	20	19	1526411	>15700000	>215828	>481464	55045	2996	3611
10	17	47	143800	1609022	38846	86350	25814	1475	1971

Table 1: Illustration of the performance impact of each of the algorithm components on a small set of moderate size instances (5 processors of capacity 100, processes weights drawn uniformly in $\{1, \ldots, 40\}$).

5. Because this quantity is measured at the end of the execution of the algorithm it provides only an order of magnitude. This is so because the algorithm tries to remove dominated triplets from a list each time a new triplet is added, as explained in Section 5.5.

6.3 Computational results

In order to reasonably explore the (practically relevant part of the) problem space we have used the scheme of Section 6.1 to generate a set of 10 instances for each $|U| \in \{2, \ldots, 14\}$ [6], each $W \in \{10, 20, \ldots, 90, 100\}$ and $C = 100$. Hence a total of 1300 instances, amongst which only 1020 were considered of nontrivial size (from around 10 up to 254 moves). For each of these sets of 10 instances, Table 2 indicates the average problem size (i.e., the average number of moves), denoted $\overline{|M|}$, as well as the number of instances in the set that the algorithm has been able to solve in less than 20 minutes, denoted n. Additionally, Table 3 provides for each value of $|U|$, the size of the biggest instance the algorithm was able to solve in less than 20 minutes, the size of the smallest instance the algorithm was *not* able to solve in less than 20 minutes as well as the size of the biggest instance on which the algorithm was tried.

Our intent, in performing this experiment, has been to obtain an idea, when the capacity constraints are extremely tight, on the kind of instances which are within the reach of the algorithm in a relatively short time for practically relevant values of $|U|$.

In the range $5 \leq |U| \leq 12$ the algorithm is able to solve most instances of size below or slightly above 40, generally in a fairly small fraction of the 20 minute limit. In this range, the algorithm is also able to solve a bunch of fairly big instances, culminating in the resolution of an instance with 11 processors and 190 moves in a bit more than 3 minutes.

Instances in the range $2 \leq |U| \leq 4$ appear to be more difficult. This is presumably due to the fact that the difficulty ends up concentrated among the few processors. As an example, for $|U| = 2$, the algorithm failed to solve an instance with 22 moves and took a bit more than 7 minutes to solve another instance with only 20 moves.

Also, in the range $13 \leq |U| \leq 14$, instances with extremely high cost optimal solutions start to appear. The algorithm seems to have difficulties in dealing with these instances as it failed to close a few relatively small instances (see Table 3) or required an important fraction of the allowed 20 minutes to solve a few other such instances. As an example, an instance with 13 processors and 24 moves was solved in a bit more than 8 minutes, this instance required the interruption of nearly 16% of the moved payload. Having said that, the practical relevance of these instances may be challenged as systematically having instances with high cost optimal solutions would be a con against embedding a reconfiguration procedure such as the present one within the design of a system. At the end of the day, what really matters is whether or not the amount of payload usually impacted by the reconfiguration is acceptable (typically below a few percent).

Lastly, it should be emphasized that when W is small enough (typically less

6. The choice for the values of $|U|$ is motivated by the fact that the system to which this work is to be applied contains at least 2 and at most 14 processors (Sirdey et al., 2003).

	$\|U\|$	2		3		4		5		6		7		8	
W		$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n
10		17.3	9	37.3	1	54.4	4	73.1	2	86.8	4	110.1	4	125.8	2
20		8.2	10	19.5	10	26.7	9	35.0	4	46.7	6	56.5	4	64.1	2
30		6.4	10	12.9	10	19.5	10	23.9	10	30.4	9	37.2	8	44.6	3
40				9.9	10	12.5	10	19.3	10	22.9	10	28.1	9	33.9	8
50						10.6	10	12.9	10	19.5	10	22.0	10	25.9	10
60								13.2	10	14.6	10	18.1	10	21.4	9
70										13.3	10	15.2	10	18.6	10
80												11.9	10	15.4	10
90														12.9	10

	$\|U\|$	9		10		11		12		13		14			
W		$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n	$\overline{\|M\|}$	n		
10		150.1	2	159.2	0	179.5	3	198.5	0	215.8	0	237.6	0		
20		75.6	3	82.1	5	92.5	1	102.6	0	111.1	0	122.4	0		
30		47.2	5	56.7	2	64.6	2	71.2	1	77.5	0	80.6	0		
40		37.3	7	45.7	3	48.0	4	51.6	3	56.8	3	58.6	2		
50		30.1	8	33.5	8	37.8	5	41.8	4	43.7	5	53.0	0		
60		25.8	9	29.5	6	29.2	8	31.8	8	35.3	6	40.8	1		
70		22.1	9	23.2	9	25.7	8	28.1	9	32.2	4	36.3	4		
80		17.3	10	19.0	10	21.2	9	25.1	9	25.5	10	28.4	5		
90		16.3	10	18.9	9	20.8	10	23.6	10	22.8	8	26.4	7		
100		12.8	10	15.8	10	17.9	10	18.2	10	19.6	9	22.7	9		

Table 2: Average instance size, denoted $\overline{|M|}$, and number of instances solved in less than 20 minutes, denoted n, for each of the 10 instances sets generated.

$\|U\|$	2	3	4	5	6	7	8	9	10	11	12	13	14
A	20	36	60	71	88	116	124	149	80	190	65	53	58
B	22	34	31	34	39	33	26	25	24	25	31	25	26
C	22	46	60	78	101	121	139	157	165	190	213	232	254

Table 3: For each value of $|U|$, row "A" indicates the size of the biggest instances solved by the algorithm in less than 20 minutes, row "B" the size of the smallest instance *not* solved by the algorithm in less than 20 minutes and row "C" provides the size of the biggest instance on which the algorithm was tried.

than or equal to 30), small cost solutions almost always exist and can be found by the algorithm, generally within a small fraction of the 20 minute limit. For example, with $|U| = 14$ and $W = 10$, the algorithm terminated with solutions situated, on average, at less than 1.2% from an hypothetical zero cost solution (given a solution of value z, distance to optimality was measured using the ratio $d(z) = \frac{z-\mathrm{OPT}}{S-\mathrm{OPT}}$, where OPT and $S = \sum_m c_m$ respectively denote the value of an optimal solution and of the worst possible one, which simply consists in interrupting all the moves [7], when unknown OPT was replaced by a lower bound e.g., 0). Overall, on the set of instances with $W \leq 30$ which the algorithm failed to solved in less than 20 minutes, solutions situated, on average, at 2.07% from an hypothetical zero cost solution were obtained.

Overall, 659 of the 1020 "hard" instances have been solved.

7 Conclusion

In this paper, we have introduced the Process Move Programming problem which consists, starting from an arbitrary initial process distribution on the processors of a distributed system, in finding the least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint is that the capacity of the processors must not be exceeded during the reconfiguration. This problem has applications in the design of high availability real-time distributed switching systems such as the one discussed in Sirdey et al. (2003).

We have shown that the PMP problem is NP-hard in the strong sense and exhibited some polynomial special cases, the most notable of which being the homogeneous case where all the processes have a constant consumption in a unique resource.

We have proposed a branch-and-bound algorithm for the general case. From an industrial perspective, it can be considered that the PMP problem is solved by this algorithm as it is able to close virtually all practical instances within a few seconds. Additionally, we have performed computational experiments demonstrating the algorithm's perspective when used to solve instances significantly harder than those occurring in practice, in terms both of size and tightness of the capacity constraints. Indeed, our algorithm was able to solve more than 64% of our such test instances within a 20 minute limit, including some instances with

7. This measure is quite natural as $1 - d(z)$ can be interpreted either as a *differential approximation ratio* (recall that differential approximation is concerned with how far the value of a solution is from the worst possible value, see Demange & Paschos, 1996) or as a *conventional approximation ratio* (Garey & Johnson, 1979) for the maximization problem complementary to the PMP problem which asks to maximize the sum of the costs of the moves which are *not* interrupted.

more than 100 moves. Also, our experiments suggest that the truncated version of the algorithm has fairly reasonable heuristic capabilities.

Nevertheless, our branch-and-bound procedure is not destined to be embedded in a real-time system. This is so mainly because the behaviour of such an algorithm may be quite sensitive to changes in the kind of instances it is asked to solve. Hence, the main purpose of our algorithm is to allow building a database of instances with known optimal solutions so as to empirically assess the quality of the solution obtained using efficient approximate resolution algorithms suitable for use in a real-time context. Efficient approximate resolution algorithms for the PMP problems are presently discussed in Sirdey et al. (2009).

Acknowledgements

The authors wish to thank the anonymous referee for several suggestions that led to improvements in the paper.

Approximate resolution of a resource-constrained scheduling problem *†

Renaud Sirdey[a] [b], Jacques Carlier[b] and Dritan Nace[b]

[a] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.
[b] UMR CNRS Heudiasyc (Université de Technologie de Compiègne), Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France.

Submitted on January 5, 2006
Revised on December 6, 2006
Accepted on March the 29th, 2007

Abstract

This paper is devoted to the approximate resolution of a strongly NP-hard resource-constrained scheduling problem which arises in relation to the operability of certain high availability real time distributed systems. We present an algorithm based on the simulated annealing metaheuristic and, building on previous research on exact resolution methods, extensive computational results demonstrating its practical ability to produce acceptable solutions, in a precisely defined sense. Additionally, our experiments are in remarkable agreement with certain theoretical properties of our simulated annealing scheme. The paper is concluded by a short discussion on further research.

Keywords: combinatorial optimization, scheduling, simulated annealing, distributed systems, OR in telecommunications.

Introduction

In this paper, we present a simulated annealing-based approximate resolution algorithm for the *Process Move Programming* (PMP) problem. This problem arises in relation to the operability of certain high-availability distributed switching systems. For example (Sirdey et al., 2003), consider a telecom switch managing radio cells on a set of call processing modules, hereafter referred to

∗. This research was supported in part by Association Nationale de la Recherche Technique grant CIFRE-121/2004.

†. Nortel GSM Access R&D Technical Report PE/BSC/INF/016550 V01/EN, appeared in the Journal of Heuristics 15:1-17, 2009.

as *processors*, of finite capacity in terms of erlangs, CPU, memory, ports, etc.; each radio cell being managed by a dedicated process running on some processor. During network operation, some cells may be dynamically added, modified (transreceivers may be added or removed) or removed, potentially leading to unsatisfactory resource utilisation in the system. This issue is addressed by first obtaining a better system configuration and by subsequently reconfiguring the system, without violation of the capacity constraints on the processors.

We now proceed with a formal definition of the problem.

Let us consider a distributed system composed of a set U of *processors* and let R denote the set of *resources* they offer. For each processor $u \in U$ and each resource $r \in R$, $C_{u,r} \in \mathbb{N}$ denotes the amount of resource r offered by processor u. We are also given a set P of applications, hereafter referred to as *processes*, which consume the resources offered by the processors. The set P is sometimes referred to as the *payload* of the system. For each process $p \in P$ and each resource $r \in R$, $w_{p,r} \in \mathbb{N}$ denotes the amount of resource r which is consumed by process p. Note that neither $C_{u,r}$ nor $w_{p,r}$ vary with time. Also, when $|R| = 1$, $C_{u,r}$ and $w_{p,r}$ are respectively denoted C_u and w_p (this principle is applied to other quantities throughout this paper).

An *admissible state* for the system is defined as a mapping $f : P \longrightarrow U \cup \{u_\infty\}$, where u_∞ is a dummy processor having infinite capacity, such that for all $u \in U$ and all $r \in R$ we have

$$\sum_{p \in P(u;f)} w_{p,r} \leq C_{u,r}, \tag{1}$$

where $P(u; f) = \{p \in P : f(p) = u\}$. The processes in $\bar{P}(f) = P(u_\infty; f)$ are not instantiated, when this set is non empty the system is in *degraded mode*.

An instance of the *Process Move Programming* (PMP) problem is then specified by two arbitrary system states f_i and f_t and, roughly speaking, consists in, starting from state f_i, finding the least disruptive sequence of operations at the end of which the system is in state f_t. The two aforementioned system states are respectively referred to as the *initial system state* and the *final system state* or, for short, the *initial state* and the *final state*[1].

Figure 1 provides an example of an instance of the PMP problem for a system with 10 processors, one resource and 46 processes. The capacity of each of the processors is equal to 30 and the sum of the consumptions of the processes is 281. The top and bottom figures respectively represent the initial and the final system states. For example, process number 23 must be moved from processor 2 to processor 6.

A process may be moved from one processor to another in two different ways:

1. Throughout the rest of this paper, it is assumed that $\bar{P}(f_i) = \bar{P}(f_t) = \emptyset$. When this is not true the processes in $\bar{P}(f_t) \setminus \bar{P}(f_i)$ should be stopped before the reconfiguration, hence some resources are freed, the processes in $\bar{P}(f_i) \setminus \bar{P}(f_t)$ should be started after the reconfiguration and the processes in $\bar{P}(f_i) \cap \bar{P}(f_t)$ are irrelevant.

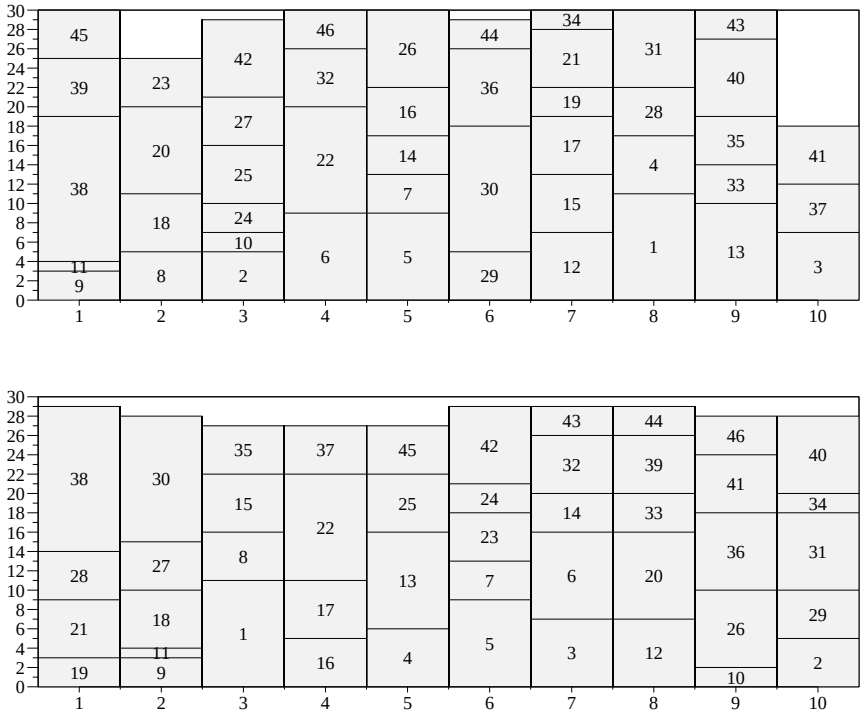


Figure 1: Example of an instance of the PMP problem.

either it is *migrated*, in which case it consumes resources on both processors for the duration of the migration and this operation has virtually no impact on service, or it is *interrupted*, that is removed from the first processor and later restarted on the other one. Of course, this latter operation has an impact on service. Additionally, it is required that the capacity constraints (1) are always satisfied during the reconfiguration and that a process is moved (i.e., migrated or interrupted) at most once. This latest constraint is motivated by the fact that a process migration is far from being a lightweight operation (for reasons related to distributed data consistency which are out of the scope of this paper), as a consequence, it is desirable to avoid processes hopping around processors.

Throughout this paper, when it is said that a move is *interrupted*, it is meant that the process associated to the move is interrupted. This slightly abusive terminology significantly lightens our discourse. Additionally, it is now assumed that $|R| = 1$, unless otherwise stated.

For each processor u, a process p in $P(u; f_i) \setminus P(u; f_t)$ must be moved from u to $f_t(p)$. Let M denote the set of process moves. Then for each $m \in M$, w_m, s_m and t_m respectively denote the amount of resource consumed by the process moved by m, the processor from which the process is moved that is the *source* of the move and the processor to which the process is moved that is the *target* of the move. Lastly, $S(u) = \{m \in M : s_m = u\}$ and $T(u) = \{m \in M : t_m = u\}$.

A pair (I, σ), where $I \subseteq M$ and where $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ is a bijection, defines an admissible *process move program*, if provided that the moves in I are interrupted (the interruptions are performed at the beginning) the other moves can be performed according to σ without inducing any violation of the capacity constraints (1). Formally, (I, σ) is an admissible program if for all $m \in M \setminus I$ we have

$$w_m \leq K_{t_m} + \sum_{\substack{m' \in I \\ s_{m'} = t_m}} w_{m'} + \sum_{\substack{m' \in S(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'}, \tag{2}$$

where $K_u = C_u - \sum_{p \in P(u; f_i)} w_p$, thereby guaranteeing that the intermediate states are admissible.

Also note that because the final state is admissible, we have, for each processor $u \in U$

$$K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u)} w_m \geq 0. \tag{3}$$

Let c_m denote the cost of interrupting m, the PMP problem then formally consists, given a set of moves, in finding a pair (I, σ) such that $c(I) = \sum_{m \in I} c_m$ is minimum.

In Sirdey et al. (2007) we have shown that the PMP problem is strongly NP-hard (even for a system with only two processors and only one resource), exhibited some polynomially solvable special cases (the most notable being $|R| = 1$

and $w_m = \text{const}$ for all $m \in M$) as well as proposed a branch-and-bound algorithm for the general case. This paper focuses on an approximate resolution algorithm based on the simulated annealing metaheuristic. Section 1 is dedicated to the theoretical considerations at the basis of our algorithm: we introduce the notion of (α, β)-acceptable solution, where β is a measure of distance to optimality and α is the probability that it is achieved, and use the markovian theory underlying the homogeneous simulated annealing algorithm to derive conditions under which such solutions may be produced. Based on these considerations, a simulated annealing-based pseudopolynomial time approximation algorithm for the PMP problem is presented in Section 2. Building on results obtained using the aforementioned branch-and-bound algorithm, we provide in Section 3 extensive computational results demonstrating the practical relevance of the method in the special case where $c_m = w_m$ (this variant still is strongly NP-hard).

1 SA-based differential approximation

Simulated annealing is a popular approximate resolution algorithm design paradigm independently introduced in the mid eighties by Kirkpatrick et al. (1983) and Cerny (1985). The main advantages of this paradigm are that it leads to relatively simple algorithms and that it is reasonably well understood from a theoretical point of view.

Throughout this section, we consider a combinatorial optimization problem which consists, given a finite set $\Omega = \{\omega_1, \ldots, \omega_N\}$ and an *objective function* $c : \Omega \longrightarrow \{e_1, \ldots, e_P\}$, in looking for an element $\omega^\star \in \Omega$ such that $e_1 = c(\omega^\star) \leq c(\omega) \leq e_P$ for all $\omega \in \Omega$. Also, a *neighbourhood function* $V : \Omega \longrightarrow 2^\Omega$ is given.

Algorithm 2 states the simulated annealing algorithm in a fairly general form. T_0 is the *initial temperature* and f (usually [2]) is a nonincreasing function referred to as the *cooling schedule*.

For background on the simulated annealing method, the reader is referred to the seminal book by van Laarhoven & Aarts (1987).

1.1 Markovian model of the SA algorithm

As noted by Aarts & van Laarhoven (1985) as well as by Lundy & Mees (1986) the behavior of the simulated annealing algorithm at temperature T can be described by means of a finite homogeneous Markov chain [3] with *transition*

2. A few authors such as Hajek & Sasaki (1989) and Möbius et al. (1997) consider cooling schedules in which the temperature is allowed to increase.

3. The reader unfamiliar with the theory of finite Markov chains is referred to Kemeny & Snell (1960).

$T \leftarrow T_0$.
Choose ω uniformly in Ω and do $\omega^\star \leftarrow \omega$.
While the stopping criterion is not satisfied do:
  Choose ω' uniformly in $V(\omega)$.
  Choose u uniformly in $[0,1]$.
  If $u \leq e^{-\frac{c(\omega')-c(\omega)}{T}}$ then [a]
    $\omega \leftarrow \omega'$.
    If $c(\omega) < c(\omega^\star)$ then $\omega^\star \leftarrow \omega$.
  End.
  $T \leftarrow f(T)$.
End.

a. Note that $e^{-\frac{c(\omega')-c(\omega)}{T}} \geq 1$ when $c(\omega') \leq c(\omega)$.

Algorithm 2: General form of the simulated annealing algorithm.

matrix

$$A_{ij}(T) = \begin{cases} 0 & \text{if } \omega_j \notin V(\omega_i), \\ \frac{1}{|V(\omega_i)|} & \text{if } \omega_j \in V(\omega_i) \text{ and } c(\omega_j) \leq c(\omega_i), \\ \frac{e^{\frac{c(\omega_i)-c(\omega_j)}{T}})}{|V(\omega_i)|} & \text{if } \omega_j \in V(\omega_i) \text{ and } c(\omega_j) > c(\omega_i), \\ 1 - \sum_{j:\omega_j \in V(\omega_i)} A_{ij}(T) & \text{if } i = j. \end{cases}$$

Under the assumption that it is both *regular* (i.e., the directed graph induced by the neighbourhood function is strongly connected) and *aperiodic*, the above chain admits a unique *stationary distribution* given by (recall that $N = |\Omega|$)

$$\pi_i^{(\infty)}(T) = \frac{e^{-\frac{c(\omega_i)}{T}}}{\sum_{j=1}^{N} e^{-\frac{c(\omega_j)}{T}}}. \tag{4}$$

Also,

$$\lim_{T\to 0} \pi_i^{(\infty)}(T) = \lim_{T\to 0} \frac{1}{\sum_{j=1}^{N} e^{\frac{c(\omega_i)-c(\omega_j)}{T}}} = \begin{cases} 0 & \text{if } c(\omega_i) > e_1, \\ \frac{1}{|\Omega^\star|} & \text{otherwise.} \end{cases}$$

1.2 Probabilistic performance guarantees

We now take the differential approximation theory point of view [4].

4. Differential approximation theory is based on the notion of differential approximation ratio which measures how far the value of a solution is from the worst possible value. Its theoretical properties are investigated in Demange & Paschos (1996) (see also Monnot et al., 2003).

Définition 1 *A solution $\omega \in \Omega$ is β-acceptable if*

$$c(\omega) \leq e_1 + \beta(e_P - e_1)$$

and (α, β)-acceptable if

$$\text{Prob}(c(\omega) \leq e_1 + \beta(e_P - e_1)) \geq \alpha,$$

where $\beta \in [0, 1]$.

Under the assumptions of regularity and aperiodicity, the following proposition provides a temperature value suitable for the production of (α, β)-acceptable solutions, given any upper bound on e_1.

Proposition 1 *Let $e_1 \leq z \leq e_P$, solutions drawn from the stationary distribution at temperature*

$$T_f(z) = \frac{\beta(e_P - z)}{\log N - \log(1 - \alpha)} \tag{5}$$

are (α, β)-acceptable.

Proof. Let $\xi = e_1 + \beta(e_P - e_1)$ then

$$\begin{aligned} P(c(\omega) > \xi; T) &= \sum_{i:e_i > \xi}^{P} \frac{N(i) e^{-\frac{e_i}{T}}}{K(T)} \\ &\leq \frac{e^{-\frac{\xi}{T}}}{K(T)} \underbrace{\sum_{i:e_i > \xi} N(i)}_{\leq N} \\ &\leq N e^{-\frac{\beta e_P}{T}} e^{\frac{\beta e_1}{T}} \underbrace{\frac{e^{-\frac{e_1}{T}}}{K(T)}}_{\leq 1} \\ &\leq N e^{-\frac{\beta(e_P - z)}{T}}. \end{aligned}$$

Where $N(i)$ is the number of solutions with value e_i and where

$$K(T) = \sum_{j=1}^{N} e^{-\frac{c(\omega_j)}{T}}.$$

Letting $N e^{-\frac{\beta(e_P - z)}{T}} = 1 - \alpha$ leads to equation (5). □

Let z_k denote the value of the best solution encountered up to iteration k, then the algorithm may be stopped as soon as $T \leq T_f(z_k)$. In other words, the

above proposition allows to use the decreasing sequence of values of the best-so-far solution so as to generate an increasing sequence of *final temperature* values, given by equation (5), and to stop the algorithm as soon as it reaches the highest of these temperatures, that is *the better the best-so-far solution, the earlier the termination*.

In general, the choice of α and β leads the algorithm into the realm of relatively small temperature values. Hence, the ability for the simulated annealing algorithm to produce (α, β)-acceptable solutions depends on how well it is able to simulate the stationary distribution at such temperature values. In order to do so, the idea consists in starting the algorithm at a relatively high temperature, say T_0, where convergence towards the stationary distribution is extremely fast, and to decrease the temperature in such a way that the stationary distributions for two succeeding values are close to each other. In particular (Aarts & van Laarhoven, 1985), choosing

$$T_{k+1} = \frac{T_k}{1 + \frac{\log(1+\delta)}{e_P+1} T_k} \tag{6}$$

guarantees that

$$|\pi_i^{(\infty)}(T_k) - \pi_i^{(\infty)}(T_{k+1})| \leq \delta,$$

where δ is a small positive real number. As a consequence, it is reasonable to expect that after decreasing the temperature only a few iterations are required in order to approach the new stationary distribution. Needless to emphasize that, despite of its reasonableness, this argument is of heuristic nature.

Note that the markovian model of Section 1.1 has inspired many other cooling schedules. See Triki et al. (2005) for a recent survey.

2 Application to the PMP problem

2.1 Approximation measure

Provided that this paper is devoted to the study of approximate resolution algorithms for the PMP problem, an approximation measure is required. Let c denote the value of the solution obtained using such an algorithm, the following approximation ratio shall be used to assess its quality

$$\beta = \frac{c - c^\star}{\sum_{m \in M} c_m - c^\star},$$

where $c^\star$ is the value of an optimal solution and where $\sum_{m \in M} c_m$ is the value of the worst possible solution which consists in interrupting all the moves.

This measure is quite natural as far as the PMP problem is concerned. On one hand, $1 - \beta$ can be interpreted as a *differential approximation ratio* (see Section

1.2). On the other hand, it can also be seen as a *conventional approximation ratio* (Garey & Johnson, 1979) for the maximization problem complementary to the PMP problem which asks to maximize the sum of the costs of the moves which are *not* interrupted.

2.2 Statement of the algorithm

As far as the PMP problem is concerned, Ω is the set of the $|M|!$ permutations of the moves and two such permutations are neighbours if one can be obtained from the other by exchanging the positions of only two moves and vice versa. It is obvious that the directed graph induced by such a neighbourhood function is strongly connected since any desired permutation can be obtained by starting with all elements in lexicographic order and then exchanging appropriate pairs of elements.

This, along with the theoretical aspects covered in the previous sections, leads to Algorithm 3. Its parameters are α, β and δ as well as the chain length (i.e., the number of iterations for each value of the temperature) which has been fixed to $|M|$ (a pragmatic as well as quite conventional choice). Also recall that for the PMP problem, the worst possible solution consists in interrupting all the moves, hence $e_P = \sum_{m \in M} c_m$ [5].

2.3 Building admissible solutions

Algorithm 3 requires a mechanism able to associate an admissible solution, hence a value, to any permutation $\pi : M \longrightarrow \{1, \ldots, |M|\}$.

First, let us remark that using the naive algorithm which tries to perform the moves in the order induced by π, interrupting those which are not feasible (see Algorithm 4), may lead to miss all optimum solutions. Figure 2 provides an example of an instance for which this happens. Indeed, the unique optimal solution clearly consists in interrupting move f and in then performing the other moves in the order *eadcb*. So either π orders move f before the other moves in which case Algorithm 4 performs it (since it is feasible) and later has to interrupt another move or π first orders a move other than f and since no such move is initially feasible the algorithm interrupts it. Both cases lead only to solution with value greater than 1, hence non optimal.

Algorithm 5 avoids this pitfall by starting from the worst solution, that is the solution which interrupts all the moves, and by trying to avoid these interruptions in the order specified by π. The principle of the algorithm is as follows. Initially all of the moves are interrupted. At the end of the k^{th} step of the loop, $I^{(k)}$ (the content of variable I at this point of the execution) contains all the moves in

5. Note that the calculation of the worst value is not always as straightforward as here and may even be as hard as finding the optimum value. See the discussion in Demange & Paschos (1996).

$T \leftarrow \sum_{m \in M} c_m$.
Initialise π to an arbitrary permutation of the moves.
Do $c \leftarrow c(\pi)$, $\pi^\star \leftarrow \pi$ and $c^\star \leftarrow c$.
While $T \geq \frac{\beta\left(\sum_{m \in M} c_m - c^\star\right)}{\log |M|! - \log(1-\alpha)}$ do:
    For $k = 1$ to $|M|$ do:
        Choose m uniformly in M and m' uniformly in $M \setminus \{m\}$.
        Exchange $\pi(m)$ and $\pi(m')$ and do $c' \leftarrow c(\pi)$.
        Choose u uniformly in $[0, 1]$.
        If $u \leq e^{-\frac{c'-c}{T}}$ then [a]
            $c \leftarrow c'$.
            If $c < c^\star$ then do $\pi^\star \leftarrow \pi$ as well as $c^\star \leftarrow c$.
        Else
            Exchange $\pi(m)$ and $\pi(m')$.
        End.
    End.
    $T \leftarrow \frac{T}{1 + \frac{\log(1+\delta)}{1+\sum_{m \in M} c_m} T}$.
End.

a. Again, note that $e^{-\frac{c(\omega')-c(\omega)}{T}} \geq 1$ when $c(\omega') \leq c(\omega)$.

Algorithm 3: Simulated annealing applied to the PMP problem.

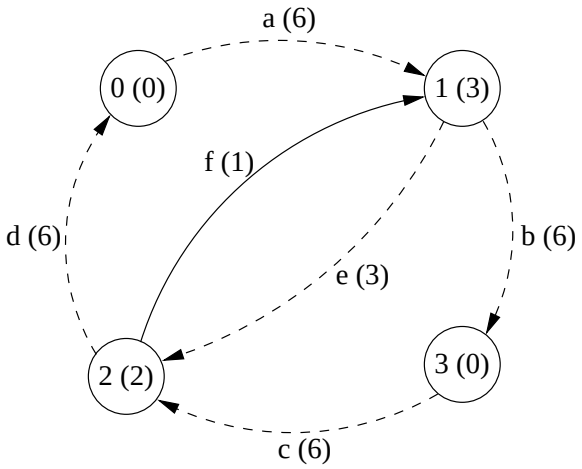


Figure 2: An instance for which using Algorithm 4 leads to miss the unique optimal solution. The number in parenthesis indicates either the initial capacity of a processor (e.g. $K_1 = 3$) or the weight of a move (e.g. $w_a = 6$).

Do $I \leftarrow \emptyset$ and $L_u \leftarrow K_u$ for all $u \in U$.
For $i = 1$ to $|M| - 1$ do:
  $m \leftarrow \pi^{-1}(i)$.
  If $L_{t_m} \geq w_m$ then
    $L_{t_m} \leftarrow L_{t_m} - w_m$.
    $L_{s_m} \leftarrow L_{s_m} + w_m$.
  Else
    $L_{s_m} \leftarrow L_{s_m} + w_m$.
    $I \leftarrow I \cup \{m\}$.
  End.
End.

Algorithm 4: A naive algorithm which builds an admissible process move program from a permutation $\pi : M \longrightarrow \{1, \ldots, |M|\}$ of the moves. At the end of the algorithm the set I contains the interrupted moves ($c(\pi) = \sum_{m \in I} c_m$), then $m \notin I$ is performed before $m' \notin I$ if $\pi(m) < \pi(m')$.

$\{m \in M : \pi(m) > k\}$ as well as potentially some of the moves not in this set, then $m \notin I^{(k)}$ is performed before $m' \notin I^{(k)}$ if $\pi(m) < \pi(m')$. Let $m = \pi^{-1}(k+1)$, the interruption of m can then be avoided (i.e., m can be removed from I) as long as no capacity constraint is violated if the process associated to m remains on s_m during the execution of the solution obtained at the end of the k^{th} step of the loop that is if

$$w_m \leq \min_{i=1,\ldots,k} \left(K_u + \sum_{m' \in S(u) \cap I^{(k)}} w_{m'} + \sum_{\substack{m' \in S(u) \setminus I^{(k)} \\ \pi(m') \leq i}} w_{m'} - \sum_{\substack{m' \in C(u) \setminus I^{(k)} \\ \pi(m') \leq i}} w_{m'} \right) = \ell_{s_m}^{(k)}, \tag{7}$$

where $\ell_{s_m}^{(k)}$ denotes the content of variable ℓ_{s_m} at the end of the k^{th} step of the loop i.e., the residual capacity which is always available on s_m during the first k steps of the loop.

Proposition 2 implies that m is then feasible.

Proposition 2 *Let (I, σ) denote an admissible process move program, if the process associated to a move $m \in I$ can remain on s_m during the entire execution of (I, σ) then, after its execution, t_m has enough remaining capacity to host the process associated to m i.e., m is possible.*

Proof. After performing all the moves in $M \setminus I$ the remaining capacity on t_m is

Do $I \leftarrow M$.
Do $L_u \leftarrow K_u + \sum_{m \in S(u)} w_m$ and $\ell_u = L_u$ for all $u \in U$.
For $i = 1$ to $|M|$ do:
  $m \leftarrow \pi^{-1}(i)$.
  If $\ell_{s_m} \geq w_m$ then
    $\ell_{s_m} \leftarrow \ell_{s_m} - w_m$.
    $L_{t_m} \leftarrow L_{t_m} - w_m$.
    $\ell_{t_m} \leftarrow \min(\ell_{t_m}, L_{t_m})$.
    $I \leftarrow I \setminus \{m\}$.
  End.
End.

Algorithm 5: Construction of an admissible process move program from a permutation $\pi : M \longrightarrow \{1, \ldots, |M|\}$ of the moves. At the end of the algorithm the set I contains the interrupted moves ($c(\pi) = \sum_{m \in I} c_m$), then $m \notin I$ is performed before $m' \notin I$ if $\pi(m) < \pi(m')$.

equal to

$$K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} - \sum_{m' \in C(t_m) \setminus I} w_{m'}$$

and, from equation (3), we have

$$K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} - \sum_{m' \in C(t_m) \setminus I} w_{m'} \geq \sum_{m' \in C(t_m) \cap I} w_{m'} \geq w_m.$$

□

Table 1 illustrates that Algorithm 5 is indeed able to build the unique optimal solution to the instance shown in Figure 2.

Lastly, it should be emphasized that Algorithm 5 can be extended to the multiple resource case in a straightforward manner.

2.4 Complexity

The following proposition quantifies the number of plateaux of temperature met by Algorithm 3.

Proposition 3 *Algorithm 3 meets* $O\left(\frac{|M| \log |M| \sum_{m \in M} c_m}{\log(1+\delta)}\right)$ *plateaux of temperature.*

U	0	1	2	3	U	0	1	2	3	m
$\ell^{(0)}$	6	12	9	6	$L^{(0)}$	6	12	9	6	
$\ell^{(1)}$	6	9	6	6	$L^{(1)}$	6	12	6	6	e
$\ell^{(2)}$	0	6	6	6	$L^{(2)}$	6	6	6	6	a
$\ell^{(3)}$	0	6	0	6	$L^{(3)}$	0	6	6	6	d
$\ell^{(4)}$	0	6	0	0	$L^{(4)}$	0	6	0	6	c
$\ell^{(5)}$	0	0	0	0	$L^{(5)}$	0	6	0	0	b
$\ell^{(6)}$	0	0	0	0	$L^{(6)}$	0	6	0	0	f

Table 1: Trace of Algorithm 5 when executed on permutation *eadcbf* of the moves of the instance shown in Figure 2. At step 6, move f is added to I since $0 = \ell_1 < w_f = 1$.

Proof. Let $\gamma = \frac{\log(1+\delta)}{1+\sum_{m\in M} c_m}$, it is easy to see that the algorithm cooling schedule (equation (6)) is such that

$$T_{k+l} = \frac{T_k}{1 + l\gamma T_k}.$$

Since at worst the algorithm remains stuck with a solution which interrupts all but one move and since such a solution has value at most $\sum_{m\in M} c_m - 1$, the number of plateaux, say Λ, is the solution of

$$\frac{T_0}{1 + \Lambda\gamma T_0} = \frac{\beta}{\log|M|! - \log(1-\alpha)},$$

that is

$$\begin{aligned}
\Lambda &= \frac{\left(1 + \sum_{m\in M} c_m\right)\left(\log|M|! - \log(1-\alpha)\right)}{\beta \log(1+\delta)} - \frac{1}{\gamma \sum_{m\in M} c_m} \\
&\propto \frac{\log|M|! \sum_{m\in M} c_m}{\log(1+\delta)} \\
&\approx \frac{(|M|-1)\log|M| \sum_{m\in M} c_m}{\log(1+\delta)}.
\end{aligned}$$

□

Since Algorithm 5 clearly runs in $O(|M|)$ and provided that $|M|$ iterations are performed for each value of the temperature, Algorithm 3 runs in

$$O\left(\frac{|M|^3 \log|M| \sum_{m\in M} c_m}{\log(1+\delta)}\right).$$

Algorithm 3 is therefore pseudopolynomial.

Despite of this result, it should be emphasized that the method is relatively computationally expensive as, for example, the constant hidden in the O-notation

has an order of magnitude of about 200 when $\beta = 0.05$ and $\delta = 0.1$. Of course, it is reasonable to expect that the worst case behavior occurs rarely in practice, as the algorithm usually finds reasonably good solutions fairly quickly (recall the discussion in Section 1.2).

3 Computational experiments

In this section, we report on computational experiments carried out so as to assess the practical ability of our simulated annealing algorithm to *to produce* 5%*-acceptable solutions (*$\beta = 0.05$*) around 95 times out of 100 (*$\alpha = 0.95$*)*, which reflects "reasonable" quality expectations. Also, δ was set to 0.1. These experiments have been performed on a Sun Ultra 10 workstation with a 440 MHz Sparc microprocessor, 512 Mo of memory and the Solaris 5.8 operating system.

3.1 Instance generation

Given U the set of processors, C the processor capacity and W an upper bound on the process consumption, an instance is generated as follows.

First, the set of processes is built by drawing consumptions uniformly in $\{1, \ldots, W\}$ until $\sum_{p \in P} w_p \geq C|U|$. The initial state, f_i, is then generated by randomly assigning the processes to the processors: the processor to which a process is assigned is drawn uniformly from the set of processors for which the remaining capacity is sufficient (note that not all processes necessarily end up assigned to a processor). The final state, f_t, is built in the very same way to the exception that only the processes which are assigned to a processor in the initial state are considered. An instance is considered valid only if all the processes assigned to a processor in the initial state are also assigned to a processor in the final state. Invalid instances are discarded and the construction process is repeated until a valid instance is obtained (the rejection rate depends on the parameters, as an example, coarse estimates for $|U| = 10$, $C = 100$ as well as $W = 10$ and $W = 50$ respectively are 29% and 41%). The set of moves is then built as explained in the introduction.

It should be emphasized that the above scheme generates instances for which the capacity constraints are extremely tight, instances which can be expected to be hard and, in particular, significantly harder than those occurring in practice. As an example, for $|U| = 10$, $C = 100$ and $W = 10$ only 1.28% of free capacity remains, on average, on each of the processors. However, for the system to which this work is to be applied (Sirdey et al., 2003) the maximum *theoretical* load of a processor ranges (nonlinearly) from at most 50% (for a system with 2 processors) to at most around 93% (for a system with 14 processors, which is the maximum). This is so because some spare capacity is provisioned for fault tolerance purpose and this spare capacity is spread among all the processors. Additionally,

it should be stressed that the system carries at most 100 processes and that a preprocessing technique, based on the fact that the properties of a system state are invariant by a permutation of the processors, is used to decrease the number of moves by around 25% on average. It turned out that our simulated annealing algorithm was able to solve virtually all practical instances to optimality and that, as a consequence, we had to consider more aggressive instance generation schemes, such as the above, in order to fairly evaluate the performances of the algorithm. As an example, on a set of 10 real instances of maximum practical size (72.3 moves, on average, for a system with 14 processors), the average time to optimality was 15.98 seconds.

Lastly, we have supposed that $c_m = w_m$, which is quite natural for our application as it is reasonable to assume that the amount of service provided by a process is proportional to the amount of resources it consumes.

3.2 Computational results

In order to reasonably explore the (practically relevant part of the) problem space we have used the scheme of Section 3.1 to generate a set of 10 instances for each $|U| \in \{2, \ldots, 14\}$ [6], each $W \in \{10, 20, \ldots, 90, 100\}$ and $C = 100$. Hence a total of 1300 instances, amongst which only 1020 were considered of nontrivial size (up to 254 moves) and used in our experiments.

In fact, the instance base is the same that we used in order to assess the practical relevance of the branch-and-bound algorithm presented in Sirdey et al. (2007), the advantage being that the value of an optimum solution is known for many of these instances.

Just running the algorithm and hoping for the best is not entirely satisfactory for two main reasons:

1. It does not give any idea as to whether or not a 5%-acceptable solution has effectively been obtained, even if we have good reasons to believe it is often so.
2. It may induce unjustified computation time as, quite often, a 5%-acceptable solution is obtained fairly early.

In order to address the above issues we proceed in two steps:

1. A lower bound, denoted by l, is obtained by solving a linear programming relaxation of the problem using a cutting plane algorithm [7] (the details of which being out of the scope of this paper, see Sirdey & Kerivin (2006a).

6. The choice for the values of $|U|$ is motivated by the fact that the system to which this work is to be applied contains at least 2 and at most 14 processors (Sirdey et al., 2003).

7. Although this relaxation involves exponentially many inequalities it can *theoretically* be solved in pseudopolynomial time. This follows from the pseudopolynomiality of the separation problem for these inequalities and from the well-known equivalence between optimization and separation (Schrijver, 1986).

2. The simulated annealing algorithm is started and stopped as soon as a solution of value less than or equal to $0.95l + 0.05 \sum_m w_m$ is encountered, if this happens.

Figure 3 provides the execution time of the algorithm for each of the instances in the base.

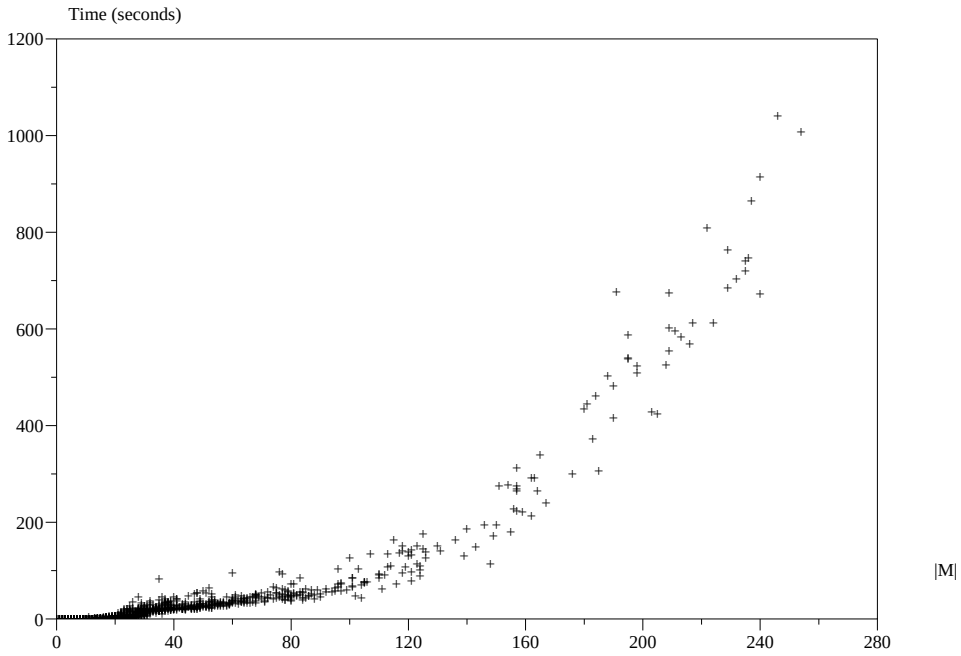


Figure 3: Execution time of the algorithm in terms of the instance size.

On the overall instance base, the algorithm was able to conclude that a 5%-acceptable solution was obtained for 868 instances (85.01%). A large part of the 152 instances for which the algorithm was not able to reach this conclusion were located in the small number of moves and small number of processors area, where the LP bound seems to be of lesser quality. This is illustrated in Figure 4. Moreover, the algorithm effectively obtained a 5%-acceptable solution for all but 23 of these 152 instances but was unable to prove it (we could reach this conclusion because we know the value of an optimal solution for many of these 152 instances). For all but 3 of these remaining 23 instances, all that was required to obtain a 5%-acceptable solution was to restart the simulated annealing step 1.7 times, on average. Finally, 10 trials of the simulated annealing step were not enough to provably obtain a 5%-acceptable solution in the case of only 3 instances, the characteristics of which being given in Table 2. Since the value of an optimal solution is unknown for these 3 instances we are unable to conclude,

although it should be emphasized that the best solutions obtained are nearly 5%-acceptable.

$\|U\|$	$\|M\|$	OPT	best β
13	36	?	5.65%
12	37	?	5.32%
10	24	?	5.90%

Table 2: Characteristics of the 3 instances for which 5%-acceptable solutions were not provably obtained, even after 10 trials of the simulated annealing step.

Figure 4 provides the repartition of the 152 instances for which the algorithm was not able to conclude that a 5%-acceptable solution was obtained. A "+" indicates an instance for which the algorithm effectively obtained a 5%-acceptable solution but was unable to prove it, a "×" indicates an instance for which more than one run of the simulated annealing step were required to obtain a 5%-acceptable solution and a "◊" indicates an instance for which no 5%-acceptable solution has provably been obtained, even after 10 trials of the simulated annealing step.

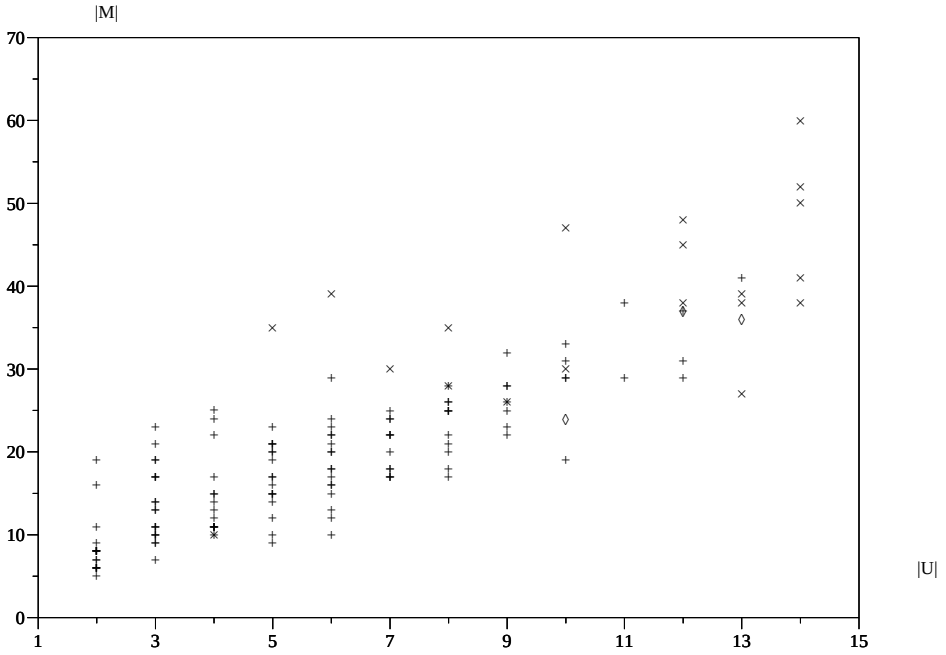


Figure 4: Repartition of the 152 instances for which the algorithm was not able to conclude that a 5%-acceptable solution was obtained.

Additionally, Table 3 illustrates the approximation performances of the algorithm by providing the percentage of instances solved within a given distance to optimality *in reality* (e.g. 1.76% of the instances were solved within a distance of 5 to 6% to optimality) and, to be fair, from the viewpoint of a "blind" user who has no knowledge on the value of an optimal solution other than provided by the lower bound. As already emphasized, the majority of the problematic instances (for the "blind" user) are located within the small number of moves (say less than 20) and small number of processors (say less than 7) area and are attributable to the relative weakness of the lower bound in this area.

	$\leq 5\%$	$]5\%, 6\%]$	$]6\%, 7\%]$	$]7\%, 8\%]$	$]8\%, 9\%]$	$]9\%, 10\%]$	$]10\%, 15\%]$	$]15\%, 18\%]$
Real	97.74%	1.76%	0.49%	0%	0%	0%	0%	0%
Blind	85.01%	4.61%	3.63%	2.06%	2.25%	0.88%	1.27%	0.20%

Table 3: Percentage of instances solved within a given distance to optimality *in reality* and from the viewpoint of a "blind" user.

If we ignore the above 3 problematic instances and coarsely estimate the probability for the simulated annealing scheme to effectively reach a 5%-acceptable solution (ratio of the number of successes over the number of trials) we obtain 0.9676, in remarkable agreement with the real value of α. Lastly, taking the pessimistic viewpoint and putting the 3 problematic instances back into the picture, that is considering 30 more trials and no additional success, leads to 0.9408, recall however that the question as to whether or not a 5%-acceptable solution has effectively been obtained is left opened for these 3 instances.

4 Conclusion

In this paper, we have proposed a simulated annealing-based approximation algorithm for the Process Move Programming problem, a strongly NP-hard scheduling problem which consists, starting from an arbitrary initial process distribution on the processors of a distributed system, in finding the least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint is that the capacity of the processors must not be exceeded during the reconfiguration. This problem has applications in the design of high availability real-time distributed switching systems such as the one discussed in Sirdey et al. (2003).

We have introduced the notion of (α, β)-acceptable solution, where β is a measure of distance to optimality and α is the probability that it is achieved, and used the markovian theory underlying the homogeneous simulated annealing algorithm to derive conditions under which such solutions may be produced. These

results have then been used to design a pseudopolynomial time approximation algorithm for the PMP problem.

Although, at the end of the day, our reasoning is heuristic, since there is no theoretical guarantee that sufficient proximity to the stationary distribution is achieved at end of each plateau of temperature, we have performed extensive computational experiments which suggest that the algorithm meets its design intent.

In these experiments, the algorithm was set up to provide $(95\%, 5\%)$-acceptable solutions. Building on previous research on exact resolution algorithms (Sirdey et al., 2007), we have been able to demonstrate that the algorithm effectively obtained a 5%-acceptable solution for 97.74% of the 1020 instances (with up to 254 moves) on which it was tried. This, as well as the analysis of the number of retrials required to obtain a 5%-acceptable solution on the remaining instances, strongly suggests that the algorithm is indeed able to produce $(95\%, 5\%)$-acceptable solutions to the PMP problem.

Despite of the above, it should be emphasized that our simulated annealing algorithm is relatively computationally expensive. Faster, although having less sound theoretical foundations, algorithms are presently discussed in Sirdey et al. (2010).

Polyhedral combinatorics of a resource-constrained ordering problem part I

On the partial linear ordering polytope *†

Renaud Sirdey[a] [b] and Hervé Kerivin[c]

[a] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.
[b] UMR CNRS Heudiasyc (Université de Technologie de Compiègne), Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France.
[c] UMR CNRS Limos (Université de Clermont-Ferrand II), Complexe scientifique des Cézeaux, 63177 Aubière Cedex, France.

September 19, 2006

Abstract

This paper is the first of a series of two devoted to the polyhedral study of a strongly NP-hard resource-constrained scheduling problem, referred to as the *process move programming problem*. This problem arises in relation to the operability of certain high-availability real time distributed systems. After a brief introduction to the problem as well as a summary of previous results, we formulate it as an integer linear program using linear ordering variables. We then drop the capacity constraints and introduce the *partial linear ordering polytope*, defined as the convex hull of all incidence vectors of arc sets of linear orderings of a node subset of the complete digraph on n nodes (the nodes not in the subset being looped), study its basic properties as well as show several classes of inequalities to be facet-defining. The companion paper is devoted to the study of the *process move program polytope* which is obtained when the capacity constraints are put back into the picture.

1 Introduction

Let us consider a distributed system composed of a set U of *processors* and let R denote the set of *resources* they offer. For each processor $u \in U$ and each

∗. This research was supported in part by Association Nationale de la Recherche Technique grant CIFRE-121/2004.
†. Technical report Nortel GSM Access R&D PE/BSC/INF/017912 V01/EN.

resource $r \in R$, $C_{u,r} \in \mathbb{N}$ denotes the amount of resource r offered by processor u. We are also given a set P of applications, hereafter referred to as *processes*, which consume the resources offered by the processors. The set P is sometimes referred to as the *payload* of the system. For each process $p \in P$ and each resource $r \in R$, $w_{p,r} \in \mathbb{N}$ denotes the amount of resource r which is consumed by process p. Note that neither $C_{u,r}$ nor $w_{p,r}$ vary with time. Also, when $|R| = 1$, $C_{u,r}$ and $w_{p,r}$ are respectively denoted C_u and w_p (this principle is applied to other quantities throughout this paper).

An *admissible state* for the system is defined as a mapping $f : P \longrightarrow U \cup \{u_\infty\}$, where u_∞ is a dummy processor having infinite capacity, such that for all $u \in U$ and all $r \in R$ we have

$$\sum_{p \in P(u;f)} w_{p,r} \leq C_{u,r}, \tag{1}$$

where $P(u; f) = \{p \in P : f(p) = u\}$. The processes in $\bar{P}(f) = P(u_\infty; f)$ are not instantiated and, when this set is non empty, the system is in *degraded mode*.

An instance of the *Process Move Programming* (PMP) problem is then specified by two arbitrary system states f_i and f_t and, roughly speaking, consists of, starting from state f_i, finding the least disruptive sequence of operations (e.g., process migrations) at the end of which the system is in state f_t. The two aforementioned system states are respectively referred to as the *initial system state* and the *final system state* or, for short, the *initial state* and the *final state*[1].

A process may be moved from one processor to another in two different ways: either it is *migrated*, in which case it consumes resources on both processors for the duration of the migration and this operation has virtually no impact on service, or it is *interrupted*, that is, removed from the first processor and later restarted on the other one. Of course, this latter operation has an impact on service. Additionally, it is required that the capacity constraints (1) are always satisfied during the reconfiguration and that a process is moved (i.e., migrated or interrupted) at most once. This latest constraint is motivated by the fact that a process migration is far from being a lightweight operation (for reasons related to distributed data consistency which are out of the scope of this paper, see e.g. Jalote, 1994) and, as a consequence, it is desirable to avoid processes hopping around processors.

Throughout this paper, when it is said that a process move is *interrupted*, it is meant that the process associated to the move is interrupted. This slightly abusive terminology significantly lightens our discourse. Additionally, it is now assumed that $|R| = 1$, unless otherwise stated.

For each processor u, a process p in $P(u; f_i) \setminus P(u; f_t)$ must be moved from

1. Throughout the rest of this paper, it is assumed that $\bar{P}(f_i) = \bar{P}(f_t) = \emptyset$. When this is not the case the processes in $\bar{P}(f_t) \setminus \bar{P}(f_i)$ should be stopped before the reconfiguration, hence some resources are freed, the processes in $\bar{P}(f_i) \setminus \bar{P}(f_t)$ should be started after the reconfiguration and the processes in $\bar{P}(f_i) \cap \bar{P}(f_t)$ are irrelevant.

u to $f_t(p)$. Let M denote the set of process moves. Then for each $m \in M$, w_m, s_m and t_m respectively denote the amount of resource consumed by the process moved by m, the processor from which the process is moved, that is, the *source* of the move and the processor to which the process is moved, that is, the *target* of the move. Lastly, $S(u) = \{m \in M : s_m = u\}$ and $T(u) = \{m \in M : t_m = u\}$.

A pair (I, σ), where $I \subseteq M$ and $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ is a bijection, defines an admissible *process move program*, if provided that the moves in I are interrupted (for operational reasons, the interruptions are performed at the beginning) the other moves can be performed, through migrations, according to σ without inducing any violation of the capacity constraints (1). Formally, (I, σ) is an admissible program if for all $m \in M \setminus I$ we have

$$w_m \leq K_{t_m} + \sum_{\substack{m' \in I \\ s_{m'} = t_m}} w_{m'} + \sum_{\substack{m' \in S(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'}, \tag{2}$$

where $K_u = C_u - \sum_{p \in P(u;f_i)} w_p$, thereby guaranteeing that the intermediate states are admissible.

Also note that because the final state is admissible, we have for each processor $u \in U$

$$K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u)} w_m \geq 0. \tag{3}$$

Let c_m denote the cost of interrupting $m \in M$. The PMP problem then formally consists, given a set of moves, of finding a pair (I, σ) such that $c(I) = \sum_{m \in I} c_m$ is minimum.

Figure 1 provides an example of an instance of the PMP problem for a system with 10 processors, one resource and 46 processes. The capacity of each of the processors is equal to 30 and the sum of the consumptions of the processes is 281. The top and bottom figures respectively represent the initial and the final system states. For example, process number 23 must be moved from processor 2 to processor 6.

In Sirdey et al. (2007) we have shown that the PMP problem is strongly NP-hard, exhibited some polynomially solvable special cases (the most notable one being $|R| = 1$ and $w_m = w$ for all $m \in M$) as well as proposed a "combinatorial" branch-and-bound algorithm for the general case. Also, an extensive literature survey was provided in that paper. Additionally, approximate resolution algorithms have been proposed in Sirdey et al. (2009) (simulated-annealing-based approach) as well as in Sirdey et al. (2010) (Grasp-based approach). This paper and its companion focus on the study of the PMP problem from the point of view of polyhedral combinatorics. In Section 2 we formulate the problem as an integer linear program. In Section 3, we introduce the *partial linear ordering polytope*, denoted P^n_{PLO}, which is obtained when the capacity constraints of the PMP problem are relaxed, study its basic properties (dimension and simple facets) as

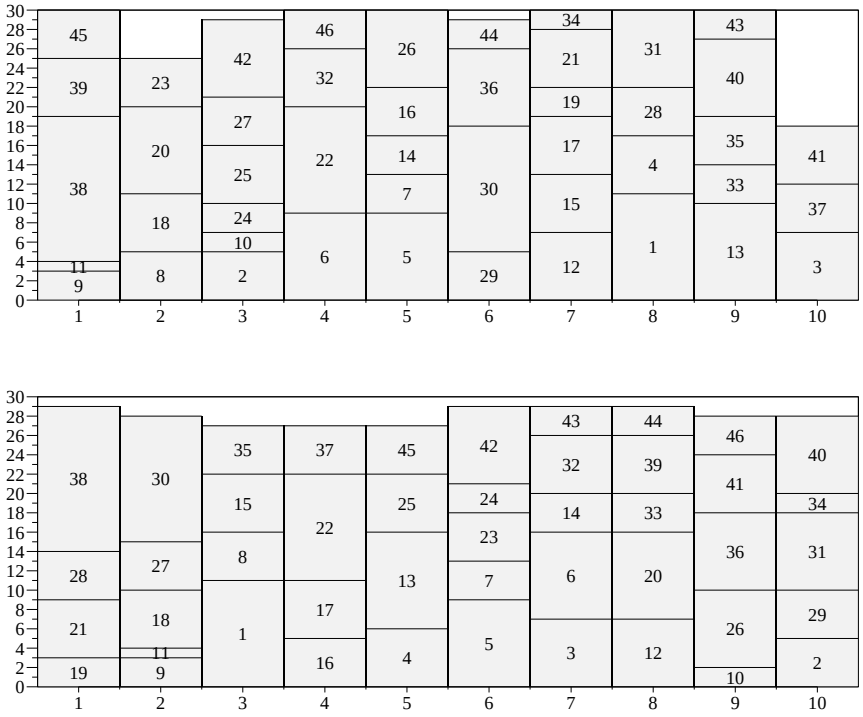


Figure 1: Example of an instance of the PMP problem.

well as demonstrate a useful trivial lifting lemma. In Section 4 we exhibit several classes of facet-defining inequalities for P^n_{PLO}, namely the k-clique, k-unicycle and k-l-bicycle inequalities, and discuss the associated separation problems.

2 An integer linear programming formulation

In this section, we formulate the PMP problem as an integer linear program. We first focus on obtaining a formulation for the decision problem which asks whether or not there exists an admissible process move program of the form $(\emptyset, \sigma)$, that is, an admissible program of zero cost. We subsequently refine the model to encompass the notion of interruption.

For each ordered pair of distinct moves of M, say m and m', we introduce the *linear ordering variables* (Queyranne & Schulz, 1994)

$$\delta_{mm'} = \begin{cases} 1 & \text{if } m \text{ precedes } m', \\ 0 & \text{otherwise.} \end{cases}$$

In order for these variables to define a valid ordering, it is natural to ask for the following constraints to be satisfied

$$\begin{cases} \delta_{mm'} + \delta_{m'm} = 1 & \forall \{m, m'\} \subseteq M, \quad (4) \\ \delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M. \quad (5) \end{cases}$$

Constraints of type (4) simply express that either m precedes m' or m' precedes m. Constraints of type (5) are known as the *transitivity constraints* and simply state that if m precedes m' and if m' precedes m'' then m precedes m''. Along with the constraints

$$\delta_{mm'} \in \{0, 1\} \qquad m \neq m' \in M, \quad (6)$$

constraints of types (4) and (5) describe a *linear ordering polytope*, that is, the convex hull of the incidence vectors of the linear orderings of the moves in M (Grötschel et al., 1985a ; Fishburn, 1992).

Since interruptions are (so far) disallowed, constraints of type (2) can be expressed as follows

$$w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} \delta_{m'm} - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} \qquad \forall m \in M. \quad (7)$$

It follows that any integral solution to the linear system of inequalities defined by the sets of constraints (4), (5), (6) and (7) (should such a solution exists) provides an admissible process move program of zero cost.

We now turn to the PMP problem and start by, for each move $m \in M$, introducing the variables

$$\delta_{mm} = \begin{cases} 1 & \text{if } m \text{ is interrupted,} \\ 0 & \text{otherwise.} \end{cases}$$

Constraints of type (2) can then be written as follows

$$(1-\delta_{mm})w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'}(\delta_{m'm'} + \delta_{m'm}) - \sum_{m' \in T(t_m)\setminus\{m\}} w_{m'}\delta_{m'm}, \quad (8)$$

for all $m \in M$. The transitivity constraints (5) remain unchanged and constraints of type (4) must be replaced by constraints

$$\delta_{mm'} + \delta_{m'm} = 1 - \max(\delta_{mm}, \delta_{m'm'}) \qquad \forall\{m, m'\} \subseteq M. \quad (9)$$

These constraints simply express that if either m or m' is interrupted then neither m precedes m' nor m' precedes m. (Recall that the interruptions are performed at the beginning.)

Proposition 1 *Constraints of type* (9) *are equivalent to the following set of constraints*

$$\begin{cases} \delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall\{m, m'\} \subseteq M, \quad (10) \\ \delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M. \quad (11) \end{cases}$$

Proof. Left to the reader. □

For reasons which will soon become obvious, inequalities of types (10) and (11) are respectively referred to as the 2-*clique* and 1-*unicycle* inequalities.

The resulting integer linear program for the process move programming problem is given Figure 2. The polytope associated to this program is hereafter referred to as the *PMP polytope* and denoted P^M_{PMP}.

Another interesting approach to the problem, which is not studied in this paper, consists, interruptions not being allowed, of minimizing the maximum overflow occurring during the reconfiguration. That is, to minimize ε, subject to constraints (4), (5) and (6) along with constraints

$$w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'}\delta_{m'm} - \sum_{m' \in T(t_m)\setminus\{m\}} w_{m'}\delta_{m'm} + \varepsilon \qquad \forall m \in M.$$

3 The partial linear ordering polytope

3.1 Definition and basic properties

The partial linear ordering polytope, denoted P^n_{PLO}, is obtained from P^M_{PMP} by dropping the capacity constraints (8). Hence, it is defined as the convex hull of all incidence vectors of arc sets of linear orderings of a node subset of the complete digraph on n nodes, the nodes not in the subset being looped. Figure 3 provides an example of a point of P^6_{PLO} which orders nodes $\{1, 2, 3, 5\}$ in the

$$
\begin{cases}
\text{Minimize } \displaystyle\sum_{m \in M} c_m \delta_{mm} & \\
\text{s. t.} & \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall \{m, m'\} \subseteq M, \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M, \\
\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M, \\
(1 - \delta_{mm}) w_m \leq K_{t_m} + \displaystyle\sum_{m' \in S(t_m)} w_{m'} (\delta_{m'm'} + \delta_{m'm}) - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} & \forall m \in M, \\
\delta_{mm'} \in \{0, 1\} & m, m' \in M.
\end{cases}
$$

Figure 2: Formulation of the PMP problem as an integer linear program.

order $(3, 5, 1, 2)$, nodes 4 and 6 being excluded and, hence, provided with a loop (i.e., $\delta_{44} = \delta_{66} = 1$).

Equivalently, P^n_{PLO} can be defined as the integral hull of the polytope defined by the following sets of inequalities (recall inequalities (10), (11) and (5))

$$\begin{cases} \delta_{ij} + \delta_{ji} + \delta_{ii} + \delta_{jj} \geq 1 & 1 \leq i < j \leq n, \quad (12) \\ \delta_{ij} + \delta_{ji} + \delta_{ii} \leq 1 & i, j \in \{1, \ldots, n\}, i \neq j, \quad (13) \\ \delta_{ij} + \delta_{jk} - \delta_{ik} \leq 1 & i, j, k \in \{1, \ldots, n\}, i \neq j \neq k \neq i, \quad (14) \\ 0 \leq \delta_{ij} \leq 1 & i, j \in \{1, \ldots, n\}, i \neq j. \end{cases}$$

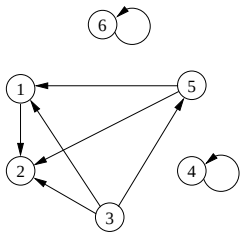


Figure 3: Example of a point of P^6_{PLO}.

P^n_{PLO} is of course interesting in its own right, it is moreover practically relevant in the sense that it can be expected that insights obtained from its study, in particular regarding its facet-defining inequalities, will be useful to gain insights regarding the structure of P^M_{PMP}.

Hereafter, in order to avoid any ambiguities, the terms *vertex* and *node* respectively refer to polytope vertices and graph vertices.

Proposition 2 *P^n_{PLO} is full-dimensional.*

Proof. P^n_{PLO} has n^2 variables, so it is required to exhibit n^2+1 affinely independent points. Let us consider the following points:

1. The unique point which orders no nodes (i.e., $\delta_{ii} = 1$ for all $i \in \{1, \ldots, n\}$).
2. The points which only order node i (i.e., $\delta_{ii} = 0$ and $\delta_{jj} = 1$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$), for each $i \in \{1, \ldots, n\}$. There are n such points.
3. The points which order only node i before node j (i.e., $\delta_{ii} = \delta_{jj} = 0$, $\delta_{kk} = 1$ for all $k \in \{1, \ldots, n\} \setminus \{i, j\}$ and $\delta_{ij} = 1$), for each $i, j \in \{1, \ldots, n\}$ with $i \neq j$. There are $n(n-1)$ such points.

It is obvious that the above n^2+1 points are linearly, hence affinely, independent. □

3.2 Trivial lifting

In this section, we prove a trivial lifting lemma which is extensively used in the sequel to shorten facet-definition proofs. Note that this kind of results is quite frequent in the domain of ordering-related polytopes (Grötschel et al., 1985a ; Müller, 1996 ; Fiorini, 2001).

Lemma 1 *Let $a^T\delta \leq \alpha$ be a facet-defining inequality for P^n_{PLO}. Setting*

$$\overline{a}_{ij} = \begin{cases} a_{ij} & \text{for all } i,j \in \{1,\ldots,n\}, \\ 0 & \text{if } i = n+1 \text{ or } j = n+1, \end{cases}$$

the inequality $\overline{a}^T\delta \leq \alpha$ then defines a facet of P^{n+1}_{PLO}.

Proof. It is obvious that $\overline{a}^T\delta \leq \alpha$ is valid for P^{n+1}_{PLO}. Let $\mathcal{F}$ be the face of P^{n+1}_{PLO} induced by $\overline{a}^T\delta \leq \alpha$. This face is clearly proper. We now need to exhibit $(n+1)^2$ affinely independent points of $\mathcal{F}$.

Since $a^T\delta \leq \alpha$ is facet-defining for P^n_{PLO}, there exist n^2 linearly independent points, say $\delta^1,\ldots,\delta^{n^2}$, so that $a^T\delta^k = \alpha$ for all $k \in \{1,\ldots,n^2\}$. Let D be the $n^2 \times n^2$ matrix having the points $\delta^1,\ldots,\delta^{n^2}$ as rows, and let $\overline{\delta}^k$ for $k = 1,\ldots,n^2$ so that

$$\overline{\delta}^k_{ij} = \begin{cases} \delta^k_{ij} & \text{for all } i,j \in \{1,\ldots,n\}, \\ 1 & \text{if } i = j = n+1, \\ 0 & \text{otherwise.} \end{cases}$$

Consider the n columns of D corresponding to the variables δ_{ii} for $i = 1,\ldots,n$. Let $\overline{D}$ be the $n^2 \times n$ matrix composed of these n columns. Since D is a nonsingular matrix, these n columns are linearly independent and therefore, there exists a $n \times n$ submatrix N of $\overline{D}$ that is nonsingular. Let u^l, for $l = 1,\ldots,n$, be the row of D corresponding to the l^{th} row of N, and let U be the $n \times n^2$ submatrix of D composed of the points u^l, for $l = 1,\ldots,n$. Let v^l, for $l = 1,\ldots,n$, be the point so that

$$v^l_{ij} = \begin{cases} u^l_{ij} & \text{for all } i,j \in \{1,\ldots,n\}, \\ 0 & \text{if } i = j = n+1, \\ 1 - u^l_{jj} & \text{if } i = n+1 \text{ and } j \in \{1,\ldots,n\}, \\ 0 & \text{if } j = n+1 \text{ and } i \in \{1,\ldots,n\}. \end{cases}$$

and w^r, for $r = 1,\ldots,n$, be the point so that

$$w^r_{ij} = \begin{cases} u^r_{ij} & \text{for all } i,j \in \{1,\ldots,n\}, \\ 0 & \text{if } i = j = n+1, \\ 0 & \text{if } i = n+1 \text{ and } j \in \{1,\ldots,n\}, \\ 1 - u^r_{ii} & \text{if } j = n+1 \text{ and } i \in \{1,\ldots,n\}. \end{cases}$$

Without loss of generality, assume that the point δ^{n^2} is different to the points u^i for $i = 1, \ldots, n$. Let $\delta^{'}$ be the point so that

$$\delta^{'}_{ij} = \begin{cases} \delta^{n^2}_{ij} & \text{for all } i,j \in \{1,\ldots,n\}, \\ 0 & \text{if } i = j = n+1, \\ 1 - u^{n^2}_{jj} & \text{if } i = n+1 \text{ and } j \in \{1,\ldots,n\}, \\ 0 & \text{if } j = n+1 \text{ and } i \in \{1,\ldots,n\}. \end{cases}$$

The points $\overline{\delta}^k$ for $k = 1, \ldots, n^2$, v^l for $l = 1, \ldots, n$, w^r for $r = 1, \ldots, n$ and $\delta^{'}$ all belong to $\mathcal{F}$. In fact, the inequality $\overline{a}^T\delta \leq \alpha$ is obviously tight for all these points. Moreover, the points v^l for $l = 1, \ldots, n$, w^r for $r = 1, \ldots, n$ and $\delta^{'}$ are obtained by ordering the node $n+1$ either before or after the nodes of $\{1, \ldots, n\}$ that are already ordered. Each row of the following $(n+1)^2 \times (n+1)^2$ matrix A corresponds to one of these $(n+1)^2$ points: the n^2 first ones correspond to the points $\overline{\delta}^k$ for $k = 1, \ldots, n^2$, the n next ones to the points v^l for $l = 1, \ldots, n$, the n next ones to the points w^r for $r = 1, \ldots, n$ and the last row to the point $\delta^{'}$.

$$A = \begin{pmatrix} D & \mathbf{0} & \mathbf{0} & e^T \\ U & \mathbf{1} - N & \mathbf{0} & 0 \\ U & \mathbf{0} & \mathbf{1} - N & 0 \\ \delta^{n^2} & e - (\delta^{n^2}_{11}, \ldots, \delta^{n^2}_{nn}) & 0 & 0 \end{pmatrix}$$

Note that the last column of A corresponds to the variable $\delta_{(n+1)(n+1)}$ and that e is the point having all its entries equal to 1.

As mentioned above, the submatrix D is nonsingular. The matrix $\mathbf{1}-N$, where $\mathbf{1}$ is the $n \times n$ matrix having all its entries equal to 1, is nonsingular since N is so. Let μ be the point corresponding to the entries $\delta^{'}_{(n+1)i}$ for $i = 1, \ldots, n$, that is, $\mu = e - (\delta^{n^2}_{11}, \ldots, \delta^{n^2}_{nn})$. By the definition of $\delta^{'}$ and u^l for $l = 1, \ldots, n$, the point μ can be written as a linear combination of the n points $(v^l_{(n+1)1}, \ldots, v^l_{(n+1)n})$, $l = 1, \ldots, n$, that is, $(\delta^{n^2}_{11}, \ldots, \delta^{n^2}_{nn}) = \sum_{l=1}^{n} \lambda_l (u^l_{11}, \ldots, u^l_{nn})$. We then can replace the last row of the matrix A by

$$\left(\delta^{n^2} - \sum_{l=1}^{n} \lambda_l u^l \qquad \mathbf{0} \qquad \mathbf{0} \qquad 0 \right).$$

We thus can deduce that the matrix A is nonsingular. We then have found $(n+1)^2$ linearly, hence affinely, independent points in $\mathcal{F}$. Therefore, $\mathcal{F}$ is a facet of P^{n+1}_{PLO}. □

3.3 Simple facets

We now turn to the study of the simple facets of P^n_{PLO}.

Proposition 3 *Inequalities $\delta_{ij} \geq 0$ $(i \neq j)$,* (12) *and* (13) *define facets of P^n_{PLO}.*

Proof. Inequalities (12) are a special case of the so-called k-clique inequalities studied in Section 4.1, hence the claim that inequalities (12) are facet-defining for P^n_{PLO} directly follows from Proposition 6.

Inequalities (13) are a special case of the so-called k-unicycle inequalities studied in Section 4.2, hence the claim that inequalities (13) are facet-defining for P^n_{PLO} directly follows from Proposition 8.

We now turn to inequality $\delta_{ij} \geq 0$ for $i, j \in \{1, \ldots, n\}$, $i \neq j$.

First, we consider the following four points: the point which orders no nodes (i.e., $\delta_{kk} = 1$ for all $k \in \{1, \ldots, n\}$), orders only node i (i.e., $\delta_{ii} = 0$ and $\delta_{kk} = 1$ for all $k \in \{1, \ldots, n\} \setminus \{i\}$), orders only node j (i.e., $\delta_{jj} = 0$ and $\delta_{kk} = 1$ for all $k \in \{1, \ldots, n\} \setminus \{j\}$), orders only node j before node i (i.e., $\delta_{ii} = \delta_{jj} = 0$, $\delta_{kk} = 1$ for all $k \in \{1, \ldots, n\} \setminus \{i, j\}$ and $\delta_{ji} = 1$). It is clear that $\delta_{ij} = 0$ for all these 4 points and that they are linearly, hence affinely, independent as for any of them there is a variable which is either 1 for the point and 0 for the others or vice-versa.

It follows that $\delta_{ij} \geq 0$ $(i \neq j)$ is a facet of $P^{|\{i,j\}|}_{\text{PLO}}$, hence a facet of P^n_{PLO} $(n \geq 2)$ by Lemma 1. □

Proposition 4 *Inequalities $\delta_{ii} \geq 0$, $\delta_{ij} \leq 1$ and* (14) *do not define facets of P^n_{PLO}.*

Proof. We first consider inequality $\delta_{ii} \geq 0$ for $i \in \{1, \ldots, n\}$. First notice that if $\delta_{ii} = 0$ for some $i \in \{1, \ldots, n\}$ then, by (12) and (13), we have $\delta_{ij} + \delta_{ji} + \delta_{jj} = 1$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$. Since P^n_{PLO} is full-dimensional we have

$$\{\delta \in P^n_{\text{PLO}} : \delta_{ii} = 0\} \subsetneq \{\delta \in P^n_{\text{PLO}} : \delta_{ij} + \delta_{ji} + \delta_{jj} = 1\} \subsetneq P^n_{\text{PLO}},$$

for any $j \in \{1, \ldots, n\} \setminus \{i\}$. Hence $\delta_{ii} \geq 0$ is not facet-defining for P^n_{PLO}.

We now turn to inequality $\delta_{ij} \leq 1$ for $i, j \in \{1, \ldots, n\}$. If $i = j$ then it is easy to see that

$$\{\delta \in P^n_{\text{PLO}} : \delta_{ii} = 1\} \subsetneq \{\delta \in P^n_{\text{PLO}} : \delta_{ik} = 0\} \subsetneq P^n_{\text{PLO}},$$

for any $k \in \{1, \ldots, n\} \setminus \{i\}$. Hence, $\delta_{ii} \leq 1$ is not facet-defining for P^n_{PLO}. If $i \neq j$, we then have

$$\{\delta \in P^n_{\text{PLO}} : \delta_{ij} = 1\} \subsetneq \{\delta \in P^n_{\text{PLO}} : \delta_{ii} = 0\} \subsetneq P^n_{\text{PLO}},$$

for all $i \in \{1, \ldots, n\}$ and all $j \in \{1, \ldots, n\} \setminus \{i\}$. Therefore, $\delta_{ij} \leq 1$ does not define a facet of P^n_{PLO}.

Lastly, since $\delta_{ij} + \delta_{jk} - \delta_{ik} = 1$ $(i \neq j \neq k \neq i)$ implies that $\delta_{jj} = 0$, we have

$$\{\delta \in P^n_{\text{PLO}} : \delta_{ij} + \delta_{jk} - \delta_{ik} = 1\} \subsetneq \{\delta \in P^n_{\text{PLO}} : \delta_{jj} = 0\} \subsetneq P^n_{\text{PLO}}$$

and, hence, the transitivity inequalities are not facet-defining for P^n_{PLO}. □

The transitivity inequalities can however be extended so as to obtain a class of facet-defining inequalities for P^n_{PLO}. This is the purpose of the following proposition.

Proposition 5 *Let $i, j, k \in \{1, \ldots, n\}$ with $i \neq j \neq k \neq i$. The extended transitivity inequality*

$$\delta_{ij} + \delta_{jk} - \delta_{ik} + \delta_{jj} \leq 1 \tag{15}$$

is a facet of P^n_{PLO}.

Proof. Validity is obvious since $\delta_{jj} = 1$ implies that $\delta_{ij} = \delta_{jk} = 0$ and since when $\delta_{jj} = 0$, inequality (15) is equivalent to a transitivity constraint (14).

We now prove that inequality (15) is indeed facet-defining for P^n_{PLO}. Let us consider the following points:

1. The point which orders no nodes (i.e., $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\}$), that is, such that $\delta_{ij} = \delta_{jk} = \delta_{ik} = 0$ and $\delta_{jj} = 1$.
2. The two points which order either only node i (i.e., $\delta_{ii} = 0$ and $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{i\}$) or only node k (i.e., $\delta_{kk} = 0$ and $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{k\}$), which are both such that $\delta_{ij} = \delta_{jk} = \delta_{ik} = 0$ and $\delta_{jj} = 1$.
3. The point which orders only node k before node i (i.e., $\delta_{ii} = \delta_{kk} = 0$, $\delta_{ll} \in \{1, \ldots, n\} \setminus \{i, k\}$ and $\delta_{ki} = 1$), that is, such that $\delta_{ij} = \delta_{jk} = \delta_{ik} = 0$ and $\delta_{jj} = 1$.
4. The two points which order either only node i before node j (i.e., $\delta_{ii} = \delta_{jj} = 0$, $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{i, j\}$ and $\delta_{ij} = 1$) or only node j before node k (i.e., $\delta_{jj} = \delta_{kk} = 0$, $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{j, k\}$ and $\delta_{jk} = 1$), which are respectively such that $\delta_{jk} = \delta_{ik} = \delta_{jj} = 0$ and $\delta_{ij} = 1$ and such that $\delta_{ij} = \delta_{ik} = \delta_{jj} = 0$ and $\delta_{jk} = 1$.
5. The point which orders only node i before node j, node i before node k and node j before node k (i.e., $\delta_{ii} = \delta_{jj} = \delta_{kk} = 0$, $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{i, j, k\}$ and $\delta_{ij} = \delta_{ik} = \delta_{jk} = 1$), that is, such that $\delta_{jj} = 0$ and $\delta_{ij} = \delta_{ik} = \delta_{jk} = 1$.
6. The point which orders only node i before node j, node k before node i and node k before node j (i.e., $\delta_{ii} = \delta_{jj} = \delta_{kk} = 0$, $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{i, j, k\}$ and $\delta_{ij} = \delta_{ki} = \delta_{kj} = 1$), that is, such that $\delta_{jj} = \delta_{ik} = \delta_{jk} = 0$ and $\delta_{ij} = 1$.
7. The point which orders only node j before node k, node k before node i and node j before node i (i.e., $\delta_{ii} = \delta_{jj} = \delta_{kk} = 0$, $\delta_{ll} = 1$ for all $l \in \{1, \ldots, n\} \setminus \{i, j, k\}$ and $\delta_{ji} = \delta_{jk} = \delta_{ki} = 1$), that is, such that $\delta_{jj} = \delta_{ij} = \delta_{ik} = 0$ and $\delta_{jk} = 1$.

It is clear that inequality (15) is tight for these 9 points of $P_{\mathrm{PLO}}^{|\{i,j,k\}|}$ and that they are linearly, hence affinely, independent. Hence, inequality (15) is facet-defining for $P_{\mathrm{PLO}}^{|\{i,j,k\}|}$ and, by Lemma 1, for P_{PLO}^{n}. □

4 Facets of the partial linear ordering polytope

Because optimizing over P_{PLO}^{n} allows to solve the linear ordering problem (simply set $c_{ii} = 0$ for all $i \in \{1, \ldots, n\}$), which is NP-hard, it is unlikely that an explicit linear description of it is ever to be obtained. Having said that, partial descriptions are known to be of the uttermost practical relevance, particularly when used within the framework of a branch-and-cut algorithm. Hence, this section is devoted to the study of several classes of facet-defining inequalities of P_{PLO}^{n}.

4.1 *k*-clique inequalities

In this section we study a class of facet-defining inequalities for P_{PLO}^{n} which we call the k-clique inequalities.

Proposition 6 *Let $I \subseteq \{1, \ldots, n\}$ with $|I| = k$, the k-clique inequality*

$$\sum_{i \in I} \sum_{j \in I} \delta_{ij} \geq |I| - 1 \tag{16}$$

is a facet of P_{PLO}^{n}.

Proof. We first prove that inequality (16) is valid for P_{PLO}^{n}. Let D_I denote the complete digraph having I as node set. In order to order m nodes in I, it is required to select at least $m-1$ arcs of D_I. Hence a point of P_{PLO}^{n} which orders m nodes in I satisfies $\sum_{i \in I} \sum_{j \in I \setminus \{i\}} \delta_{ij} \geq m - 1$ and $\sum_{i \in I} \delta_{ii} = |I| - m$. The claim of validity follows.

We now prove that inequality (16) is facet-defining for P_{PLO}^{n}. Let us consider the following points of $P_{\mathrm{PLO}}^{|I|}$:

1. The points which only order node k, for each $k \in I$ (i.e., $\delta_{kk} = 0$ and $\delta_{ll} = 1$ for all $l \in I \setminus \{k\}$). There are $|I|$ such points and it is clear that inequality (16) is tight for them.
2. The points which only order node k_1 before node k_2, for each $k_1 \in I$ and $k_2 \in I$ with $k_1 \neq k_2$ (i.e., $\delta_{k_1 k_1} = \delta_{k_2 k_2} = 0$, $\delta_{ll} = 1$ for all $l \in I \setminus \{k_1, k_2\}$ and $\delta_{k_1 k_2} = 1$). There are $|I|(|I|-1)$ such points and (again) it is clear that inequality (16) is tight for them.

Linear, hence affine, independence of the above $|I|^2$ points is straightforward. Hence, since they all belong to $P_{\mathrm{PLO}}^{|I|}$, inequality (16) is facet-defining for $P_{\mathrm{PLO}}^{|I|}$ and, as a consequence, is also facet-defining for P_{PLO}^{n} by Lemma 1. □

Figure 4 provides an example of a 4-clique inequality for P_{PLO}^{6} with $I = \{1, 2, 3, 5\}$.

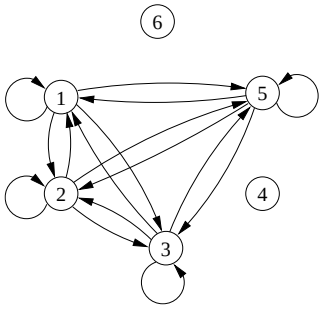


Figure 4: Example of a 4-clique on P_{PLO}^{6}.

As already stated, inequality (10) is nothing but a 2-clique inequality induced by $\{i, j\}$ and, hence, is facet-defining for P_{PLO}^{n}.

Also, note that for fixed k, the k-clique inequalities can be separated in polynomial time as there are only polynomially many (C_n^k) of them. This remark, however, is of little practical relevance.

Moreover, we have the following proposition.

Proposition 7 *The separation problem for the k-clique inequalities is NP-hard.*

Proof. It is easy to see that, given $\delta^\star \in \mathbb{R}^{n^2}$, the separation problem for the k-clique inequalities requires solving the mathematical program

$$\begin{cases} \text{Minimize } x^T \Delta x \\ \text{s. t. } x \in \{0,1\}^n, \end{cases}$$

where x is the incidence vector of set I and where Δ is a $n \times n$ matrix such that

$$\Delta_{ij} = \begin{cases} \delta_{ij}^\star & \text{if } i \neq j, \\ \delta_{ii}^\star - 1 & \text{otherwise.} \end{cases}$$

Then, consider an instance of the max-cut problem

$$\begin{cases} \text{Maximize } \displaystyle\sum_{i=1}^{n} \sum_{j=1}^{i-1} a_{ij} x_i (1 - x_j) \\ \text{s. t. } x \in \{0,1\}^n. \end{cases} \tag{17}$$

where x is the incidence vector of a node subset of the complete graph on n nodes and a_{ij} is the valuation of the edge incident to both i and j. Provided that (17) is equivalent to

$$\text{Minimize } \sum_{i=1}^{n}\sum_{j=1}^{i-1} a_{ij}x_ix_j - \sum_{i=1}^{n} x_i \sum_{j=1}^{i-1} a_{ij},$$

letting

$$\delta^{\star}_{ij} = \begin{cases} a_{ij} & \text{if } i > j, \\ 1 - \sum\limits_{k=1}^{i-1} a_{ik} & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases}$$

allows to use an algorithm solving the separation problem for the k-clique inequalities to solve the max-cut problem. Hence the claim follows. □

Such a complexity result does not diminish the potential practical relevance of the k-clique inequalities, as, in general, separation problems do not need to be solved exactly when embedded within a branch-and-cut algorithm. Furthermore, there exists high quality heuristics for unconstrained binary quadratic problems (Glover et al., 2002).

4.2 *k*-unicycle inequalities

In this section we study a class of facet-defining inequalities for P^n_{PLO} which we call the k-unicycle inequalities.

Proposition 8 *Let $I \subset \{1,\ldots,n\}$ with $|I| = k$ and $i_0 \in \{1,\ldots,n\} \setminus I$. The k-unicycle inequality*

$$\delta_{i_0i_0} + \sum_{i\in I}(\delta_{ii_0} + \delta_{i_0i}) - \sum_{i\in I}\sum_{j\in I\setminus\{i\}} \delta_{ij} \leq 1 \tag{18}$$

is a facet of P^n_{PLO}.

Proof. We first show that inequality (18) is valid for P^n_{PLO} by considering the following cases:

1. A point which neither orders i_0 nor any nodes in I (i.e., $\delta_{ii} = 1$ for all $i \in I \cup \{i_0\}$) only selects the 1-valued loop of node i_0 [2] and, hence, the left-hand side of (18) has value 1.
2. A point which orders i_0 and does not order any nodes in I (i.e., $\delta_{i_0i_0} = 0$ and $\delta_{ii} = 1$ for all $i \in I$) does not select any arcs and, as a consequence, the left-hand side of (18) has value 0.

2. Of the digraph induced by inequality (18).

3. A point which does not order i_0 and orders nodes in $\emptyset \subset J \subseteq I$ (i.e., $\delta_{i_0 i_0} = 1$, $\delta_{ii} = 0$ for all $i \in J$ and $\delta_{ii} = 1$ for all $i \in I \setminus J$) selects the 1-valued loop of node i_0 as well as $\frac{|J|(|J|-1)}{2}$ (-1)-valued arcs (in-between the nodes in J), hence, the value of the left-hand side of (18) is less than or equal to 1.
4. A point which orders i_0 as well as the nodes in $\emptyset \subset J \subseteq I$ (i.e., $\delta_{i_0 i_0} = 0$, $\delta_{ii} = 0$ for all $i \in J$ and $\delta_{ii} = 1$ for all $i \in I \setminus J$) selects $|J|$ 1-valued arcs (between i_0 and the nodes in J) as well as $\frac{|J|(|J|-1)}{2}$ (-1)-valued arcs (in between the nodes in J), as a consequence, the value of the left-hand side of (18) is equal to $-|J|^2 < 0$.

The claim of validity then follows.

We now prove that inequality (18) is facet-defining for $P^n_{\rm PLO}$. Let us consider the following points of $P^{|I \cup \{i_0\}|}_{\rm PLO}$:

1. The point which orders no nodes (i.e., $\delta_{ii} = 1$ for all $i \in I \cup \{i_0\}$) and for which inequality (18) is obviously tight.
2. The points which only order node k, for each $k \in I$ (i.e., $\delta_{kk} = 0$, $\delta_{i_0 i_0} = 1$ and $\delta_{ii} = 1$ for all $i \in I \setminus \{k\}$). There are $|I|$ such points and inequality (18) is obviously tight for them (only the 1-valued loop of node i_0 is selected). Linear independence follows from the fact that a point in this set is the only point so far such that $\delta_{kk} = 0$.
3. The points which order either only node i_0 before node k or only node k before i_0, for each $k \in I$ (i.e., $\delta_{i_0 k} = 1$ or $\delta_{k i_0} = 1$ with $\delta_{ii} = 1$ for all $i \in I \setminus \{k\}$). There are $2|I|$ such points and inequality (18) is tight for them (only one 1-valued arc is selected, in-between i_0 and k). Linear independence follows from the fact that a point in this set is the only point so far such that either $\delta_{k i_0} = 1$ or $\delta_{i_0 k} = 1$.
4. The points which order only node i_0 before node k_1, node i_0 before node k_2 and either node k_1 before node k_2 or node k_2 before node k_1, for each $\{k_1, k_2\} \subseteq I$ (i.e., $\delta_{i_0 k_1} = \delta_{i_0 k_2} = 1$, $\delta_{k_1 k_2} = 1$ or $\delta_{k_2 k_1} = 1$, and $\delta_{ii} = 1$ for all $i \in I \setminus \{k_1, k_2\}$). There are $|I|(|I|-1)$ such points and inequality (18) is tight for them (two 1-valued arcs and one (-1)-valued arc are selected). Linear independence follows from the fact that a point in this set is the only point so far such that either $\delta_{k_1 k_2} = 1$ or $\delta_{k_2 k_1} = 1$.

Hence, we have exhibited a set of $(|I|+1)^2$ linearly, hence affinely, independent points of $P^{|\{i_0\} \cup I|}_{\rm PLO}$ for which inequality (18) is tight. It follows that inequality (18) is facet-defining for $P^{|\{i_0\} \cup I|}_{\rm PLO}$ as well as for $P^n_{\rm PLO}$ by Lemma 1. □

Figure 5 provides an example of a 3-unicycle inequality for $P^6_{\rm PLO}$ with $i_0 = 1$ and $I = \{3, 4, 5\}$.

As already stated, inequality (11) is nothing but a 1-unicycle inequality with $i_0 = i$ and $I = \{j\}$. Hence, it is facet-defining for $P^n_{\rm PLO}$.

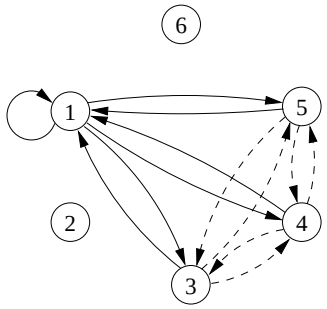


Figure 5: Example of a 3-unicycle on P^6_{PLO}. Dashed arcs are valued by -1.

Again, note that for fixed k, there are only polynomially many (nC^k_{n-1}) k-unicycle inequalities. Moreover, we have the following proposition.

Proposition 9 *The separation problem for the k-unicycle inequalities is NP-hard.*

Proof. Let $x_i = 1$, $i \in \{1, \ldots, n\}$, if and only if $i = i_0$ and let $y_i = 1$, $i \in \{1, \ldots, n\}$, if and only if $i \in I$. Given $\delta^\star \in \mathbb{R}^{n^2}$, the separation problem for the k-unicycle inequalities then requires solving the following mathematical program

$$\begin{cases} \text{Maximize } \displaystyle\sum_{i=1}^{n} \delta^\star_{ii} x_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} (\delta^\star_{ij} + \delta^\star_{ji}) x_i y_j - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \delta^\star_{ij} y_i y_j, \\ \text{s. t.} \\ \displaystyle\sum_{i=1}^{n} x_i = 1, & (19) \\ y_i \leq 1 - x_i \ \forall i \in \{1, \ldots, n\}, & (20) \\ x_i \in \{0, 1\} \ \forall i \in \{1, \ldots, n\}, \\ y_i \in \{0, 1\} \ \forall i \in \{1, \ldots, n\}, \end{cases}$$

where constraint (19) guarantees that an i_0 is chosen and where constraints (20) enforces that i_0 must not belong to I.

Then, consider an instance of the max-cut problem on the complete graph on n nodes (the notations are carried over from the proof of Proposition 7) as well as an instance of the separation problem for the k-unicycle inequalities in $\mathbb{R}^{(n+1)^2}$ where

$$\delta^\star_{ij} = \begin{cases} \delta^\star_{ij} = a_{ij} & \text{if } i < j, \\ 0 & \text{otherwise,} \end{cases}$$

for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, n\}$, where $\delta^\star_{i,n+1} = \sum_{i=1}^{j-1} a_{ij}$ and $\delta^\star_{n+1,i} = 0$ for $i \in \{1, \ldots, n\}$ as well as, finally, where $\delta^\star_{n+1,n+1}$ is set to a suitable huge value chosen in order to ensure that $i_0 = n + 1$ (i.e., that $x_{n+1} = 1$) in all optimal solutions.

This allows to use an algorithm for solving the separation problem for the k-clique inequalities on P^{n+1}_{PLO} to solve the max-cut problem on the complete graph on n nodes. Hence the claim follows. □

4.3 *k-l*-bicycle inequalities

In this section we study a class of facet defining inequalities for P^n_{PLO} which we call the k-l-bicycle inequalities.

Proposition 10 *Let $i_0 \in \{1,\dots,n\}$, $j_0 \in \{1,\dots,n\} \setminus \{i_0\}$, $\emptyset \subset I \subset \{1,\dots,n\} \setminus \{i_0, j_0\}$ and $\emptyset \subset J \subset \{1,\dots,n\} \setminus \{i_0, j_0\}$ with $|I| = k$, $|J| = l$ and $I \cap J = \emptyset$. The k-l-bicycle inequality*

$$\delta_{i_0 i_0} + \delta_{j_0 j_0} + \delta_{i_0 j_0} + \sum_{i\in I}(\delta_{i i_0} + \delta_{i_0 i} - \delta_{i j_0}) + \sum_{j\in J}(\delta_{j j_0} + \delta_{j_0 j} - \delta_{i_0 j})$$
$$- \sum_{i\in I}\sum_{i'\in I\setminus\{i\}} \delta_{ii'} - \sum_{j\in J}\sum_{j'\in J\setminus\{j\}} \delta_{jj'} - \sum_{i\in I}\sum_{j\in J} \delta_{ji} \le 2 \qquad (21)$$

is a facet of P^n_{PLO}.

Proof. We first prove that inequality (21) is valid for P^n_{PLO}. Let us consider a point which orders the moves in $I' \subseteq I$ and the moves in $J' \subseteq J$. Note that such a point necessarily selects $\frac{|I'|(|I'|-1)}{2}$ and $\frac{|J'|(|J'|-1)}{2}$ (-1)-valued arcs (in-between the nodes in I' and J', respectively). Now we prove the validity claim by considering the following cases:

1. If the point orders neither node i_0 nor node j_0 (i.e., $\delta_{i_0 i_0} = \delta_{j_0 j_0} = 1$), then the left-hand side of (21) has value $2 - \frac{|I'|(|I'|-1)}{2} - \frac{|J'|(|J'|-1)}{2} \le 2$.
2. If the point orders node i_0 and does not order node j_0 (i.e., $\delta_{i_0 i_0} = 0$ and $\delta_{j_0 j_0} = 1$), then the left-hand side of (21) has value at most $1 + |I'| - \frac{|I'|(|I'|-1)}{2} - \frac{|J'|(|J'|-1)}{2}$, that is, is less than or equal to 2 since $|I'| - \frac{|I'|(|I'|-1)}{2} \le 1$.
3. If the point orders node j_0 and does not order node i_0 (i.e., $\delta_{j_0 j_0} = 0$ and $\delta_{i_0 i_0} = 1$), then the left-hand side of (21) has value at most $1 - \frac{|I'|(|I'|-1)}{2} + |J'| - \frac{|J'|(|J'|-1)}{2}$, that is, is less than or equal to 2 for the same reason as above.
4. If the point orders both node i_0 and node j_0 (i.e., $\delta_{i_0 i_0} = \delta_{j_0 j_0} = 0$), then there is at most $1 + |I'| + |J'|$ selected 1-valued arcs ($\{i_0, j_0\}$ along with the arcs between the nodes in I' and i_0 and the arcs between the nodes in J' and j_0) along with the $\frac{|I'|(|I'|-1)}{2} + \frac{|J'|(|J'|-1)}{2}$ selected (-1)-valued arcs (in-between the nodes in I' and in-between the nodes in J'). We have $1 + |I'| - \frac{|I'|(|I'|-1)}{2} + |J'| - \frac{|J'|(|J'|-1)}{2} \le 3$, equality occurring only when both

I' and J' are of cardinality 1 or 2 and $\delta_{i_0 j_0} = 1$. Let $i \in I'$ and $j \in J'$ and assume that $\delta_{i_0 j_0} = 1$. If i precedes i_0 then since i_0 precedes j_0 the (-1)-valued arc from i_0 to j must be selected (by transitivity). If j_0 precedes j then since i_0 precedes j_0 the (-1)-valued arc from i_0 to j must be selected (by transitivity). So assume that i_0 precedes i and that j precedes j_0. We then have that either the (-1)-valued arcs between j and i is selected or both (-1)-valued arcs from i to j_0 and from i_0 to j must be selected so as to break the cycles in the digraph shown on Figure 6. Hence, at least one (-1)-valued arc is selected on top of the $\frac{|I'|(|I'|-1)}{2} + \frac{|J'|(|J'|-1)}{2}$ already identified ones. Therefore the left-hand side of (21) is less than or equal to 2.

The claim of validity then follows.

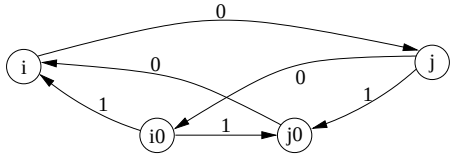


Figure 6: A non feasible ordering with $\delta_{i_0 i} = \delta_{j j_0} = \delta_{i_0 j_0} = 1$.

We now prove that inequality (21) is facet defining for P^n_{PLO}. Let us consider the following points of $P^{|I \cup J \cup \{i_0, j_0\}|}_{\text{PLO}}$:

1. The point which orders no nodes (i.e., $\delta_{ii} = 1$ for all $i \in I \cup J \cup \{i_0, j_0\}$) for which inequality (21) is obviously tight.
2. The points which only order node k for each $k \in I \cup J$ (i.e., $\delta_{kk} = 0$, $\delta_{i_0 i_0} = \delta_{j_0 j_0} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k\}$). There are $|I| + |J|$ such points and it is clear that inequality (21) is tight for them.
3. The points which only order either node k before node i_0 or node i_0 before node k for each $k \in I$ (i.e., $\delta_{k i_0} = 1$ or $\delta_{i_0 k} = 1$, $\delta_{j_0 j_0} = 1$ and $\delta_{ii} = 1$ for each $i \in (I \cup J) \setminus \{k\}$). There are $2|I|$ such points and inequality (21) is tight for them.
4. The points which only order either node k before node j_0 or node j_0 before node k for each $k \in J$ (i.e., $\delta_{k j_0} = 1$ or $\delta_{j_0 k} = 1$, $\delta_{i_0 i_0} = 1$ and $\delta_{ii} = 1$ for each $i \in (I \cup J) \setminus \{k\}$). There are $2|J|$ such points and inequality (21) is tight for them.
5. The points which only order node k_1 before node k_2 for each $k_1 \in I$ and $k_2 \in J$ (i.e., $\delta_{i_0 i_0} = \delta_{j_0 j_0} = \delta_{k_1 k_2} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). There are $|I||J|$ such points and inequality (21) is tight for them.
6. The points which only order node i_0 before node j_0, node i_0 before node k and node j_0 before node k for each $k \in I$ (i.e., $\delta_{i_0 j_0} = \delta_{i_0 k} = \delta_{j_0 k} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k\}$). There are $|I|$ such points and inequality (21) is tight for them.

7. The points which only order node k before node i_0, node k before node j_0 and node i_0 before node j_0 for each $k \in J$ (i.e., $\delta_{ki_0} = \delta_{kj_0} = \delta_{i_0j_0} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k\}$). There are $|J|$ such points and inequality (21) is tight for them.

8. Given any $k_1 \in I$ and any $k_2 \in J$, the point which only orders node j_0 before node k_1, node j_0 before node k_2, node j_0 before node i_0, node k_1 before node k_2, node k_1 before node i_0 and node k_2 before node i_0 (i.e., $\delta_{j_0k_1} = \delta_{j_0k_2} = \delta_{j_0i_0} = \delta_{k_1k_2} = \delta_{k_1i_0} = \delta_{k_2i_0} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). Inequality (21) is tight for this point.

9. The points which only order node i_0 before node k_1, node i_0 before node k_2 and node k_1 before node k_2 for each $k_1 \in I$ and $k_2 \in I \setminus \{k_1\}$ (i.e., $\delta_{i_0k_1} = \delta_{i_0k_2} = \delta_{k_1k_2} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). There are $|I|(|I| - 1)$ such points and inequality (21) is tight for them. Note that variable $\delta_{k_1k_2}$ has a -1 coefficient in inequality (21).

10. The points which only order node j_0 before node k_1, node j_0 before node k_2 and node k_1 before node k_2, for each $k_1 \in J$ and $k_2 \in J \setminus \{k_1\}$ (i.e., $\delta_{j_0k_1} = \delta_{j_0k_2} = \delta_{k_1k_2} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). There are $|J|(|J| - 1)$ such points and inequality (21) is tight for them. Note that variable $\delta_{k_1k_2}$ has a -1 coefficient in inequality (21).

11. Given any $k_1 \in J$, the points which only order node k_2 before node k_1, node k_2 before node i_0, node k_2 before node j_0, node k_1 before node i_0, node k_1 before node j_0 and node i_0 before node j_0 for each $k_2 \in I$ (i.e., $\delta_{k_2k_1} = \delta_{k_2i_0} = \delta_{k_2j_0} = \delta_{k_1i_0} = \delta_{k_1j_0} = \delta_{i_0j_0} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). There are $|I|$ such points and inequality (21) is tight for them. Note that variable $\delta_{k_2j_0}$ has a -1 coefficient in inequality (21).

12. Given any $k_1 \in I$, the points which only order node i_0 before node j_0, node i_0 before node k_1, node i_0 before node k_2, node j_0 before node k_1, node j_0 before node k_2 and node k_1 before node k_2 for each $k_2 \in J$ (i.e., $\delta_{i_0j_0} = \delta_{i_0k_1} = \delta_{i_0k_2} = \delta_{j_0k_1} = \delta_{j_0k_2} = \delta_{k_1k_2} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). There are $|J|$ such points and inequality (21) is tight for them. Note that variable $\delta_{i_0k_2}$ has a -1 coefficient in inequality (21).

13. The points which only order node k_2 before node i_0, node k_2 before node j_0, node k_2 before node k_1, node i_0 before node j_0, node i_0 before node k_1 and node j_0 before node k_1 for each $k_1 \in I$ and each $k_2 \in J$ (i.e., $\delta_{k_2i_0} = \delta_{k_2j_0} = \delta_{k_2k_1} = \delta_{i_0j_0} = \delta_{i_0k_1} = \delta_{j_0k_1} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). There are $|I||J|$ such points and inequality (21) is tight for them. Note that variable $\delta_{k_2k_1}$ has a -1 coefficient in inequality (21).

14. Given any $k_1 \in I$ and any $k_2 \in J$, the point which only orders node k_1 before node k_2, node k_1 before node i_0, node k_2 before node i_0 (i.e., $\delta_{k_1k_2} = \delta_{k_1i_0} = \delta_{k_2i_0} = \delta_{j_0j_0} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). Inequality (21) is tight for this point.

15. Given any $k_1 \in I$ and any $k_2 \in J$, the point which only orders node j_0 before node k_1, node j_0 before node k_2, node k_1 before node k_2 (i.e., $\delta_{j_0k_1} = \delta_{j_0k_2} = \delta_{k_1k_2} = \delta_{i_0i_0} = 1$ and $\delta_{ii} = 1$ for all $i \in (I \cup J) \setminus \{k_1, k_2\}$). Inequality (21) is tight for this point.

Points in the union S of the sets 1 to 13 given above belong to $P_{\text{PLO}}^{|I\cup J\cup\{i_0,j_0\}|}$ and are linearly, hence affinely, independent. This is so because for each point in S there is a variable which takes value 1 (respectively 0) and which has value 0 (respectively 1) for all the preceding points.

Furthermore, it is easy to verify that point 14 is the only one satisfying

$$\delta_{j_0j_0} + \delta_{k_1i_0} + \delta_{k_2i_0} = 3,$$

and that point 15 is the only one satisfying

$$\delta_{i_0i_0} + \delta_{j_0k_1} + \delta_{j_0k_2} = 3.$$

Therefore, we have exhibited a set of $1+(|I|+|J|)+2|I|+2|J|+|I||J|+|I|+|J|+1+|I|(|I|-1)+|J|(|J|-1)+|I|+|J|+|I||J|+1+1 = (|I|+|J|+2)^2$ affinely independent points of $P_{\text{PLO}}^{|I\cup J\cup\{i_0,j_0\}|}$. It follows that inequality (21) is facet-defining for $P_{\text{PLO}}^{|\{i_0,j_0\}\cup I\cup J|}$ and, by Lemma 1, also facet-defining for P_{PLO}^n. □

Figure 7 provides an example of a 2-3-bicycle inequality on P_{PLO}^9 with $i_0 = 7$, $j_0 = 1$, $I = \{8, 9\}$ and $J = \{3, 4, 5\}$.

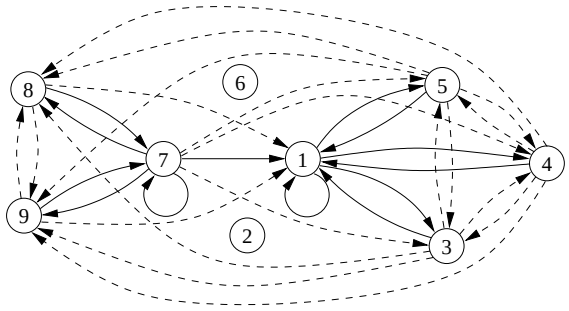


Figure 7: Example of a 2-3-bicycle on P_{PLO}^9. Dashed arcs are valued by -1.

Yet again, there are only polynomially many ($n^2 C_{n-2}^k C_{n-k-2}^l$) k-l-bicycle inequalities, for fixed k and l. Finally, we have the following proposition.

Proposition 11 *The separation problem for the k-l-unicycle inequalities is NP-hard.*

Proof. The proof is left to the reader, due to its obvious similarity with that of Proposition 9. □

5 Conclusion

In this paper, we have introduced an integer linear program formulation, based on linear ordering variables, for the *process move programming problem*, a strongly NP-hard scheduling problem which consists, starting from an arbitrary initial process distribution on the processors of a distributed system, of finding the least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint is that the capacity of the processors must not be exceeded during the reconfiguration. This problem has applications in the design of high availability real-time distributed switching systems such as the one discussed in Sirdey et al. (2003).

Ignoring the capacity constraints, as a first step, lead us to define and study the *partial linear ordering polytope*. In particular, we have introduced several classes of facet-defining inequalities for this polytope and, as we shall demonstrate in the companion paper, it turns out that they all define facets of the *process move program polytope* under mildly restrictive assumptions.

Polyhedral combinatorics of a resource-constrained ordering problem part II

On the process move program polytope *†

Hervé Kerivin[a] and Renaud Sirdey[b] [c]

[a] UMR CNRS Limos (Université de Clermont-Ferrand II), Complexe scientifique des Cézeaux, 63177 Aubière Cedex, France.
[b] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.
[c] UMR CNRS Heudiasyc (Université de Technologie de Compiègne), Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France.

September 19, 2006

Abstract

This paper is the second of a series of two devoted to the polyhedral study of a strongly NP-hard resource-constrained scheduling problem, referred to as the *process move programming problem*. In the present paper, we put back into the picture the capacity constraints which were ignored in the first paper. In doing so, we introduce the *process move program polytope*, study its basic properties and show several classes of inequalities to be facet-defining. Some of the latter were proved to be facet-defining for the *partial linear ordering polytope* which was both introduced and studied in the companion paper.

1 Introduction

Recall that the *process move programming problem* (denoted *PMP problem*) consists, starting from an arbitrary initial distribution of processes on the processors of a distributed system, of finding the least disruptive sequence of operations (i.e., non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main

∗. This research was supported in part by Association Nationale de la Recherche Technique grant CIFRE-121/2004.
†. Technical report Nortel GSM Access R&D PE/BSC/INF/017913 V01/EN.

constraints are that the capacity of the processors must not be exceeded during the reconfiguration and that a process is moved (i.e., migrated or interrupted) exactly once.

More precisely, let U and M respectively denote the set of processors and the set of process moves. Then for each $m \in M$, c_m, w_m, s_m and t_m respectively denote the cost of interrupting the process moved by m (i.e., of *interrupting* m), the amount of resource consumed by the process moved by m (i.e., the *weight* of m), the processor from which the process is moved (i.e., the *source* of m) and the processor to which the process is moved (i.e., the *target* of m). Also, K_u denotes the initial remaining capacity of processor $u \in U$, $S(u) = \{m \in M : s_m = u\}$ and $T(u) = \{m \in M : t_m = u\}$. Lastly, a pair (I, σ), where $I \subseteq M$ and $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ is a bijection, defines an admissible *process move program* if, provided that the moves in I are interrupted (for operational reasons, the interruptions are performed at the beginning of the reconfiguration), the other moves can be performed (i.e., migrated) according to σ without inducing any violation of the capacity constraints.

Let $\delta_{mm} = 1$ if and only if $m \in M$ is interrupted and $\delta_{mm'} = 1$ if and only if $m \in M$ is performed (without interruption) before $m' \in M \setminus \{m\}$ is performed (also without interruption). As shown in the companion paper (Sirdey & Kerivin, 2006c), the process move programming problem can then be expressed as the integer linear program given in Figure 1. Constraints (1), (2), (3) and (4) are respectively referred to as the 2-*clique inequalities*, the 1-*unicycle inequalities*, the *extended transitivity inequalities* and the *capacity inequalities*.

This paper is devoted to the study of the polytope associated with the above program, called the *process move program polytope* and denoted P^M_{PMP}. In Section 2, we study the basic properties of P^M_{PMP}. In particular, we establish necessary and sufficient conditions for full dimensionality as well as investigate conditions under which the inequality classes $0 \leq \delta_{mm'}$ (for $m, m' \in M$), (1), (2), (3) and (4) are facet-defining for P^M_{PMP}. In Section 3, we build on results obtained in the companion paper and investigate which of the classes of facet-defining inequalities indentified for the partial linear ordering polytope also define facets for P^M_{PMP} and under which conditions. We then subsequently introduce two classes of inequalities, the *source* and *target cover inequalities*, valid for P^M_{PMP}, and we provide both necessary and sufficient conditions for them to be facet-defining as well as pseudopolynomial separation algorithms.

2 The process move program polytope

The process move program polytope

$$P^M_{\mathrm{PMP}} = \mathrm{conv}\{\delta^{n^2} \in \mathbb{R}^{n^2} : \delta \text{ satisfies (1)-(5)}\}$$

$$
\begin{cases}
\text{Minimize } \displaystyle\sum_{m\in M} c_m\delta_{mm} & & \\
\text{s. t.} & & \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall\{m,m'\}\subseteq M, & (1)\\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m\neq m'\in M, & (2)\\
\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} + \delta_{m'm'} \leq 1 & m\neq m'\neq m''\neq m\in M, & (3)\\
(1-\delta_{mm})w_m - \displaystyle\sum_{m'\in S(t_m)} w_{m'}(\delta_{m'm'}+\delta_{m'm}) + \sum_{m'\in T(t_m)\setminus\{m\}} w_{m'}\delta_{m'm} \leq K_{t_m} & \forall m\in M, & (4)\\
\delta_{mm'}\in\{0,1\} & m,m'\in M. & (5)
\end{cases}
$$

Figure 1: Formulation of the PMP problem as an integer linear program.

where $n = |M|$, corresponds to the points of the partial linear ordering polytope P^n_{PLO} that satisfy the capacity constraints (4). Therefore we first study the necessary and sufficient conditions that make facet-defining inequalities of P^n_{PLO} remain facets of P^M_{PMP}. Remark that any valid but not facet-defining inequality of P^n_{PLO} cannot obviously define a facet of P^M_{PMP}. To do so, the following notations are introduced. Given two subsets of moves $\{m_1, \ldots, m_r\} \subseteq M$ and $X \subseteq M$ with $1 \leq r \leq n$ and $\{m_1, \ldots, m_r\} \cap X = \emptyset$ (X may be empty), a *incomplete process move program* is denoted by $[m_1, \ldots, m_r; X]$ if the only specified ordering is the one on $\{m_1, \ldots, m_r\}$, that is, the pair (I, σ) is chosen so that $I = M \setminus (\{m_1, \ldots, m_r\} \cup X)$ and $\sigma(m_i) < \sigma(m_j)$ for all $i,\ j \in \{1, \ldots, r\}$ with $i < j$. Remark that a process move program corresponds to when $X = \emptyset$. The incomplete process move program $[m_1, \ldots, m_r; X]$ is *admissible* if and only if there exists a point $\delta \in \mathbb{R}^{n^2}$ in P^M_{PMP} so that

$$\delta_{mm} = \begin{cases} 0 & \text{if } m \in \{m_1, \ldots, m_r\} \cup X, \\ 1 & \text{otherwise,} \end{cases}$$

and

$$\delta_{m_i m_j} = \begin{cases} 1 & \text{if } m_i,\ m_j \in \{m_1, \ldots, m_r\} \text{ with } i < j, \\ 0 & \text{if } m_i,\ m_j \in \{m_1, \ldots, m_r\} \text{ with } i \geq j. \end{cases}$$

Note that δ is not necesseraly unique because the imposed ordering is only on the process moves in $\{m_1, \ldots, m_r\}$. If $X = \emptyset$, the point δ is then unique and the process move program is simply denoted by $[m_1, \ldots, m_r]$.

2.1 Basic properties

Proposition 1 *Polytope P^M_{PMP} is full-dimensional if and only if $[m, m']$ is an admissible program for all $m, m' \in M$ with $m \neq m'$.*

Proof. If there exist $m, m' \in M$ with $m \neq m'$ and $[m, m']$ is not admissible, we then have $P^M_{\text{PMP}} \subseteq \{\delta \in \mathbb{R}^{n^2} : \delta_{mm'} = 0\}$. Hence $\dim P^M_{\text{PMP}} \leq n^2 - 1$.

Conversely, the admissibility of the program $[m, m']$ for all $m, m' \in M$ with $m \neq m'$ implies that the $n^2 + 1$ affinely independent vertices considered in the proof of Proposition 2 in the companion paper still belong to P^M_{PMP}. Remark that with the aim of shortening some of the proofs throughout the paper, we will refer to points of P^n_{PLO} that we used in proofs in the companion paper instead of introducing them again. □

In the sequel, the full dimensional of P^M_{PMP} is assumed, unless otherwise stated. Given $N \subseteq M$, the process move programming problem restricted to N is obtained from the one associated with M by only considering the process moves in N and by increasing the initial remaining capacity of $u \in U$ by $\sum_{m \in S(u) \cap (M \setminus N)} w_m$,

that is, by considering that the process moves in $M \setminus N$ are interrupted. Remark that the polytope P^N_{PMP} is full-dimensional as well.

Given two distinct admissible programs p and p', we say that p' is *dominated* by p if for any pair of distinct process moves $m_1, m_2 \in M$, m_1 is ordered before m_2 in p' implies m_1 is ordered before m_2 in p. This dominance relationship is denoted $p' \prec p$. Note that the set of interrupted process moves in p is included in the set of interrupted processes in p'.

Let $a^T\delta \leq \alpha$ induce a facet $\mathcal{F}$ of P^n_{PLO}. Given $m_1, m_2 \in M$ with $m_1 \neq m_2$, consider the restriction $\mathcal{F}_{m_1m_2}$ of $\mathcal{F}$ onto the subspace of $\mathbb{R}^{n^2}$ defined by $\delta_{m_1m_2} = 1$, that is, $\mathcal{F}_{m_1m_2} = \mathcal{F} \cap \{\delta \in \mathbb{R}^{n^2} : \delta_{m_1m_2} = 1\}$. Let $A_{m_1m_2}$ be the set of programs associated with the points of $\mathcal{F}_{m_1m_2}$ and so that for any program $p \in A_{m_1m_2}$, there does not exist $p' \in A_{m_1m_2} \setminus \{p\}$ with $p' \prec p$. The set $A_{m_1m_2}$ contains nothing but the programs that order m_1 before m_2 and that involve the minimum number of process moves. Note that it does not mean that all the programs in $A_{m_1m_2}$ involve the same number of moves. We can now give a necessary condition for facet-defining inequalities of P^n_{PLO} to be facet-defining for P^M_{PMP}.

Proposition 2 *Let $a^T\delta \leq \alpha$ define a nontrivial facet $\mathcal{F}$ of P^n_{PLO}. Then $a^T\delta \leq \alpha$ defines a facet of P^M_{PMP} only if for all $m_1, m_2 \in M$ with $m_1 \neq m_2$, there exists an admissible program in $A_{m_1m_2}$.*

Proof. If there exist $m_1, m_2 \in M$ so that $m_1 \neq m_2$ and no programs in $A_{m_1m_2}$ are admissible, the face of P^M_{PMP} induced by $a^T\delta \leq \alpha$ is then included in the one induced by $\delta_{m_1m_2} \geq 0$. Since $a^T\delta \leq \alpha$ is a nontrivial inequality and P^M_{PMP} is assumed to be full-dimensional, $a^T\delta \leq \alpha$ does not define a facet of P^M_{PMP}. □

Unlike what we got for the partial linear ordering polytope (see Lemma 1 in the companion paper, we do not have a trivial lifting lemma for the process move program polytope that from a facet-defining inequality of P^M_{PMP}, gives a facet-defining inequality of $P^{M\cup\{m\}}_{\text{PMP}}$, $m \notin M$, by setting zero coefficients to the variables involving m. Nevertheless, we prove two restricted lifting lemmas for the process move program polytope that will be useful to prove some of the following results. In fact using these lifting results, we will mostly need to focus on the process move subset supporting the considered inequality to prove that the latter is facet-defining for P^M_{PMP}. To make the statements of the lifting lemmas clearer (and their use as well), we introduce some additional notations. Given an inequality $a^T\delta \leq \alpha$ that defines a face $\mathcal{F}$ of P^M_{PMP}, let $M_0 = \{m_0 \in M : a_{m_0m} = a_{mm_0} = 0 \text{ for all } m \in M\}$ and $M_s = M \setminus M_0$. Remark that M_s is the process move subset that supports the inequality $a^T\delta \leq \alpha$, that is, for any $m \in M_s$, there exists a non-zero coefficient in $a^T\delta \leq \alpha$ that corresponds to a variable involving

m. Denote by $\mathcal{F}_s$ the face of $P^{M_s}_{\rm PMP}$ induced by $a^T\delta \leq \alpha$. Let

$$F_i = \{m_i \in M_s : \max\{ \sum_{m\in M_s} \delta_{mm} : \delta \in \mathcal{F}_s \text{ and } \delta_{m_i m_i} = 0\} = |M_s| - i\}$$

for $i = 1, \ldots, s$ where $s \leq n$ and $F_j = \emptyset$ for all $j \in \{s+1, \ldots, n_s\}$. The process move subsets F_i for all $\{1, \ldots, s\}$ are clearly pairwise disjoint and their union is nothing but M_s.

Proposition 3 *Let $a^T\delta \leq \alpha$ be a valid inequality of P^M_{PMP}. If $a^T\delta \leq \alpha$ is facet-defining for $P^{M_s}_{PMP}$ and the two following conditions hold*

i) the program $[\emptyset]$ induces a point of P^M_{PMP} satisfying $a^T\delta \leq \alpha$ with equality,

ii) for any $m_0 \in M_0$ and $m \in F_k$ with $k \in \{2, \ldots, s\}$, there exist $X_a \subseteq \bigcup_{h=1}^{k-1} F_h$ and $X_b \subseteq \bigcup_{h=1}^{k-1}$ so that the incomplete programs $[m_0, m; X_a]$ and $[m, m_0; X_b]$ induce points of P^M_{PMP} satisfying $a^T\delta \leq \alpha$ with equality,

then the inequality $a^T\delta \leq \alpha$ is also facet-defining for P^M_{PMP}.

Proof. Since $a^T\delta \leq \alpha$ defines a facet, say $\mathcal{F}_s$, of $P^{M_s}_{\rm PMP}$, there exist n_s^2 affinely-independent points $\delta^1, \ldots, \delta^{n_s^2}$ of $\mathcal{F}_s$, where $n_s = |M_s|$. For $i = 1, \ldots, n_s^2$, let $\overline{\delta}^i \in \{0,1\}^{n^2}$ be so that

$$\overline{\delta}^i_{mm'} = \begin{cases} \delta^i_{mm'} & \text{for all } m,\ m' \in M_s, \\ 1 & \text{if } m = m' \in M_0, \\ 0 & \text{otherwise.} \end{cases}$$

We clearly have $\overline{\delta}^i \in \mathcal{F} = \{\delta \in P^M_{\rm PMP} : a^T\delta = \alpha\}$ for $i = 1, \ldots, n_s^2$, and these n_s^2 points are affinely independent. Let m_0 and m_0' be two distinct points of M_0. Because of Condition i) and the full-dimension assumption on $P^M_{\rm PMP}$, the program $[m_0, m_0']$ clearly corresponds to a point of $\mathcal{F}$. Denote by S_0 the set of $n_0(n_0-1)$ points hence obtained, where $n_0 = |M_0|$. The sets of points $\{\overline{\delta}^i : i = 1, \ldots, n_s^2\}$ and S_0 form an affinely-independent set since any point in S_0 is the only one having a variable $\delta_{m_0 m_0'}$, $\{m_0, m_0'\} \subseteq M_0$, equal to 1. Consider now a process move $m_k \in F_k$ with $k \in \{1, \ldots, s\}$. By Condition ii), for any $m_0 \in M_0$ there exist $X_a \subseteq \bigcup_{h=1}^{k-1} F_h$ and $X_b \subseteq \bigcup_{h=1}^{k-1} F_h$ so that the incomplete programs $[m_0, m; X_a]$ and $[m, m_0; X_b]$ induce points $\overline{\delta}^{m_k}$ and $\tilde{\delta}^{m_k}$, respectively, that belong to $\mathcal{F}$. Remark that if $k = 1$, we have $X_a = X_b = \emptyset$ and from the full-dimension assumption on $P^M_{\rm PMP}$ those two points belong to $\mathcal{F}$. It is obvious that the points $\overline{\delta}^{m_k}$, $\tilde{\delta}^{m_k}$ and $\overline{\delta}^1, \ldots, \overline{\delta}^{n_s^2}$ are affinely independent. Moreover because of $X_a \cap F_h = \emptyset$ and $X_b \cap F_h = \emptyset$ for $h = k, \ldots, s$, the $2n_0 \sum_{k=1}^{s} |F_k|$ points obtained as previously

described are affinely independent. In fact, they can be ordered so that any of these points corresponds to the first one having a variable $\delta_{mm'}$ equal to 1 where $|\{m, m'\} \cap M_s| = 1$. Finally for any process move m_0 of M_0, Condition i) implies that the program $[m_0]$ induces a point δ^{m_0} that belongs to $\mathcal{F}$. Point δ^{m_0} together with all the ones we have considered so far form an affinely-independent set since it is the only one satisfying

$$\delta_{m_0 m_0} + \sum_{m \in M_s} (\delta_{m_0 m} + \delta_{m m_0}) = 0.$$

Therefore, we have obtained $n_s^2 + n_0(n_0-1) + 2n_0 \sum_{k=1}^{s} |F_k| + n_0 = n_s^2 + n_0^2 + 2n_0 n_s = n^2$ affinely-independent points of $\mathcal{F}$. Our proof is then completed. □

Before stating our second restricted lifting result, we give a technical lemma that will be useful in some of the forthcoming proofs.

Lemma 1 *Let $m_1, m_2, m_3 \in M$ with $m_1 \neq m_2 \neq m_3 \neq m_1$. We then have $P_{PMP}^M \cap \{\delta \in \{0,1\}^{n^2} : \delta_{m_1 m_2} = 1$ and $\delta_{m_3 m_3} = 0\} \neq \emptyset$.*

Proof. Let $\overline{\delta} \in X = P_{\text{PMP}}^M \cap \{\delta \in \{0,1\}^{n^2} : \delta_{m_1 m_2} = 1$ and $\delta_{m_3 m_3} = 0\}$. We first remark that if $\overline{\delta}_{mm} = 0$ for any $m \in M \setminus \{m_1, m_2, m_3\}$, then the point $\tilde{\delta}$ obtained from $\overline{\delta}$ by setting $\tilde{\delta}_{mm} = 1$, $\tilde{\delta}_{mm'} = \tilde{\delta}_{m'm} = 0$ for all $m' \in M \setminus \{m\}$ and keeping all the other components unchanged, also belongs to X. Therefore we can suppose without loss of generality that $\overline{\delta}_{mm} = 1$ for all $m \in M \setminus \{m_1, m_2, m_3\}$. Assume $t_{m_1} \neq s_{m_2}$, that is, the target processor of move m_1 is different than the source processor of move m_2. Since full dimension is assumed, the process move program $[m_1, m_3, m_2]$ is admissible. Hence, the associated point belongs to X. Suppose now that $t_{m_1} = s_{m_2}$. If $s_{m_2} = s_{m_3}$, the program $[m_3, m_1, m_2]$ is admissible. If $s_{m_2} \neq s_{m_3}$, the program $[m_1, m_2, m_3]$ is admissible. In both cases, the point associated with the program belongs to X. Therefore, $X \neq \emptyset$. □

Proposition 4 *Let $a^T\delta \leq \alpha$ be a valid inequality of P_{PMP}^M. If $a^T\delta \leq \alpha$ is facet-defining for $P_{PMP}^{M_s}$ and the two following conditions hold*

i) $F_1 = M_s$,

ii) for any $m_0 \in M_0$, there exists $X \subseteq M_s$ so that $|X| \geq 2$ and the incomplete programs $[m_0; X]$ induces a point of P_{PMP}^M satisfying $a^T\delta \leq \alpha$ with equality,

then the inequality $a^T\delta \leq \alpha$ is also facet-defining for P_{PMP}^M.

Proof. The proof is quite similar to the one of Proposition 3. In fact denoting the points in the same way, Condition i) together with the full-dimension assumption and Lemma 1 is enough to get all the points of S_0, the ones of $\{\overline{\delta}^{m_1} : m_1 \in F_1\}$ and those of $\{\tilde{\delta}^{m_1} : m_1 \in F_1\}$. These points and those of $\{\overline{\delta}^{i} : i = 1, \ldots, n_s^2\}$

clearly form an affinely-independent set. The points of $\{\delta^{m_0} : m_0 \in M_0\}$ then exist because of Condition ii). The latter are affinely independent with the first $n^2 - n_0$ ones considered so far in the proof since for any $m_0 \in M_0$, point δ^{m_0} is the only one that satisfies

$$\sum_{m \in M_s} (\delta_{m_0 m} + \delta_{m m_0}) \geq 2.$$

Therefore, the face of P^M_{PMP} induced by $a^T\delta \leq \alpha$ contains n^2 affinely-indenpendent points. □

2.2 Simple facets

We now investigate which of the inequality classes $\delta_{mm'} \geq 0$ (for $m, m' \in M$), (3) and (4) are facet-defining for P^M_{PMP}. We are postponing the study of constraints (1) and (2) to Section 3.1 because they are respectively special cases of the so-called k-clique and k-unicycle inequalities we will study later.

Proposition 5 *For $m_1, m_2 \in M$, the inequality $\delta_{m_1 m_2} \geq 0$ is facet-defining for P^M_{PMP} if and only if $m_1 \neq m_2$.*

Proof. From Proposition 4 in the companion paper, $\delta_{mm} \geq 0$ cannot define a facet of P^M_{PMP} for all $m \in M$ since it is not facet-defining for P^n_{PLO}. Assume now that $m_1 \neq m_2$. From the full-dimensional assumption, any program $[m, m']$ is admissible where $m, m' \in M$ and $m \neq m'$. Consider then the point corresponding to the interruption of all the process moves and the $n^2 - 1$ points corresponding to all the admissible programs involving exactly two process moves but the program $[m_1, m_2]$. These n^2 points are clearly affinely independent and they belong to the face of P^M_{PMP} induced by $\delta_{m_1 m_2} \geq 0$. □

Proposition 6 *Let $m_1, m_2, m_3 \in M$ with $m_1 \neq m_2 \neq m_3 \neq m_1$. The extended transitivity inequality*

$$\delta_{m_1 m_2} + \delta_{m_2 m_3} - \delta_{m_1 m_3} + \delta_{m_2 m_2} \leq 1 \tag{6}$$

is facet-defining for P^M_{PMP} if and only if the following conditions hold

i) the programs $[m_1, m_2, m_3]$, $[m_2, m_3, m_1]$ and $[m_3, m_1, m_2]$ are admissible,

ii) for any $m \notin M \setminus \{m_1, m_2, m_3\}$,

ii.a) at least one of the following programs $[m_1, m, m_2]$, $[m, m_1, m_2]$ or $[m, m_2, m_3]$ is admissible,

ii.b) at least one of the following programs $[m_1, m_2, m]$, $[m_2, m_3, m]$ or $[m_2, m, m_3]$ is admissible.

Proof. To prove the necessary condition, let us express the sets $A_{mm'}$ for all $m, m' \in M$ with $m \neq m'$ and then, let us apply Proposition 2. For $m \in M \setminus \{m_1, m_2, m_3\}$, we have

$$A_{mm'} = \begin{cases} \{[m, m']\} & \text{if } m' \in M \setminus \{m, m_2\}, \\ \{[m_1, m, m_2], [m, m_1, m_2], [m, m_2, m_3]\} & \text{if } m' = m_2. \end{cases}$$

We also have

$$A_{m_1 m'} = \begin{cases} \{[m_1, m']\} & \text{if } m' \in M \setminus \{m_1, m_3\}, \\ \{[m_1, m_2, m_3]\} & \text{if } m' = m_3, \end{cases}$$

$$A_{m_2 m'} = \begin{cases} \{[m_2, m_3, m_1]\} & \text{if } m' = m_1, \\ \{[m_2, m_3]\} & \text{if } m' = m_3, \\ \{[m_1, m_2, m], [m_2, m_3, m], [m_2, m, m_3]\} & \text{if } m' \in M \setminus \{m_2, m_1, m_3\}, \end{cases}$$

$$A_{m_3 m'} = \begin{cases} \{[m_3, m']\} & \text{if } m' \in M \setminus \{m_3, m_2\}, \\ \{[m_3, m_1, m_2]\} & \text{if } m' = m_2. \end{cases}$$

From the full-dimension assumption on P^M_{PMP}, we can deduce the admissibility of all the programs in the sets $A_{mm'}$, $\{m, m'\} \subseteq M$ involving exactly two process moves. Condition i) (respectively ii)) is obviously implied by A_{m_1,m_3}, A_{m_2,m_1} and A_{m_3,m_2} (respectively A_{mm_2} and $A_{m_2 m}$ for all $m \in M \setminus \{m_1, m_2, m_3\}$) each having at least one admissible program. Therefore, inequality (6) is facet-defining for P^M_{PMP} only if conditions i) and ii) are fulfilled.

Suppose now these two conditions are satisfied. We have $M_s = \{m_1, m_2, m_3\}$. Let $\mathcal{F}_s$ be the face of $P^{M_s}_{\text{PMP}}$ induced by (6) and let $\overline{\delta}$ be one of the 9 points introduced in the proof of Proposition 5 in the companion paper. If $\overline{\delta}$ belongs to one of the first four sets, its associated program then orders at most two moves. Since the polytope $P^{M_s}_{\text{PMP}}$ is assumed to be full-dimensional, these 6 points belong to $\mathcal{F}_s$. The admissibility of $[m_1, m_2, m_3]$, $[m_2, m_3, m_1]$ and $[m_3, m_1, m_2]$ implies that the 3 points in sets 5 to 7 also belong to $\mathcal{F}_s$. Inequality (6) then is facet-defining for $P^{M_s}_{\text{PMP}}$. Finally, it is straightforward to see that both conditions of Proposition 3 are satisfied. We clearly have $F_1 = \{m_1, m_3\}$, $F_2 = \{m_2\}$ and the program interrupting all the process moves of M corresponds to a point of the face of P^M_{PMP} induced by (6). Moreover for any $m_0 \in M_0$, Condition ii.a) (respectively ii.b)) implies there exists $\emptyset \neq X_a \subsetneq \{m_1, m_3\}$ (respectively $\emptyset \neq X_b \subsetneq \{m_1, m_3\}$) so that A_{mm_2} (respectively $A_{m_2 m}$) has an admissible program. We then deduce that inequality (6) defines a facet of P^M_{PMP}. □

Proposition 7 *Let $m \in M$. The capacity inequality*

$$(1 - \delta_{mm})w_m + \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} - \sum_{m' \in S(t_m)} w_{m'}(\delta_{m'm'} + \delta_{m'm}) \leq K_{t_m} \quad (7)$$

does not define a facet of P^M_{PMP}.

Proof. Given $m_0 \in M$, let $\mathcal{F}$ be the face of P^M_{PMP} induced by the capacity inequality (7) associated with m_0. Assume that $\mathcal{F}$ is a facet of P^M_{PMP}. From the assumption on the full dimension of polytope P^M_{PMP}, there exists $\delta^1 \in \mathcal{F}$ so that $\delta^1_{m_0m_0} = 1$. We then have $\delta^1_{m_0m} = \delta^1_{mm_0} = 0$ for all $m \in M \setminus \{m_0\}$, and (7) can be rewritten as

$$-\sum_{m \in S(t_{m_0})} w_m \delta^1_{mm} = K_{t_{m_0}}.$$

Since $w_m > 0$ for all $m \in M$ and $K_u \geq 0$ for all $u \in U$, we deduce $\delta^1_{mm} = 0$ for all $m \in S(t_{m_0})$ and $K_{t_{m_0}} = 0$.

If $S(t_{m_0}) = \emptyset$ the left-hand side of (7) is then positive for any point δ of $\mathcal{F}$ so that $\delta_{m_0m_0} = 0$. Since $K_{t_{m_0}} = 0$, we then have $\mathcal{F} \subsetneq \{\delta \in P^M_{\text{PMP}} : \delta_{m_0m_0} = 1\} \subsetneq P^M_{\text{PMP}}$. Therefore, we can suppose $S(t_{m_0}) \neq \emptyset$. Let $\delta^2 \in \mathcal{F}$ so that $\delta^2_{m_0m_0} = 0$. Such a point exists since $\mathcal{F}$ is a facet of the full-dimensional polytope P^M_{PMP}. For any $m_1 \in S(t_{m_0})$, the valid inequalities $\delta^2_{m_0m_1} + \delta^2_{m_1m_0} + \delta^2_{m_1m_1} \leq 1$ and $\delta^2_{m_0m_1} + \delta^2_{m_1m_0} + \delta^2_{m_0m_0} + \delta^2_{m_1m_1} \geq 1$ combined with $\delta^2_{m_0m_0} = 0$ give $\delta^2_{m_0m_1} + \delta^2_{m_1m_0} + \delta^2_{m_1m_1} = 1$, that is,

$$\delta^2_{m_0m_1} + \delta^2_{m_1m_0} + \delta^2_{m_0m_0} + \delta^2_{m_1m_1} = 1. \tag{8}$$

We previously have shown that (8) also holds for δ^1 and then for any point of $\mathcal{F} \cap \{\delta \in P^M_{\text{PMP}} : \delta_{m_0m_0} = 1\}$. Therefore, we obtain

$$\mathcal{F} \subsetneq \{\delta \in P^M_{\text{PMP}} : \delta_{m_0m_1} + \delta_{m_0m_0} + \delta_{m_0m_0} + \delta_{m_1m_1} = 1\} \subsetneq P^M_{\text{PMP}},$$

a contradiction. □

3 Facets of the process move program polytope

3.1 Facets from the partial linear ordering polytope

We now give necessary and sufficient conditions for facets of P^n_{PLO} introduced in the companion paper to be also facets of P^M_{PMP}. From the definition of the polytope P^M_{PMP}, any valid inequality of P^n_{PLO} is obviously valid for P^M_{PMP}. The next proposition states that the k-clique inequalities define facets of P^M_{PMP}. We remind that inequalities (1) correspond to a special case of k-clique inequalities (i.e., $k = 2$) and then, are also facet-defining for P^M_{PMP}.

Proposition 8 *Let $I \subseteq M$ with $|I| = k$. The k-clique inequality*

$$\sum_{i \in I} \sum_{j \in I} \delta_{ij} \geq |I| - 1 \tag{9}$$

is facet-defining for P^M_{PMP}.

Proof. Given $m, m' \in M$ with $m \neq m'$, we have

$$A_{mm'} = \begin{cases} \{[m, m']\} & \text{if } m \in I,\ m' \in I, \\ \{[m, m']\} & \text{if } m \in I,\ m' \notin I, \\ \{[m, m']\} & \text{if } m \notin I,\ m' \in I, \\ \{[m, m'; i] : i \in I\} & \text{if } m \notin I,\ m' \notin I. \end{cases}$$

We clearly have $M_s = I$ and $F_1 = I$. From the full-dimension assumption on P^M_{PMP} and Lemma 1, $A_{mm'}$ contains an admissible program for all $m, m' \in M$ with $m \neq m'$. Moreover for any $m_0 \in M \setminus I$ and $\{i_1, i_2\} \subseteq I$, the incomplete program $[m_0; \{i_1, i_2\}]$ is admissible and corresponds to a point satisfying (9) with equality. It is then obvious that if inequality (9) is facet-defining for P^I_{PMP}, so it is for P^M_{PMP} by Proposition 4. Let $\mathcal{F}_s$ be the face of P^I_{PMP} induced by inequality (9). Consider the $|I|^2$ points introduced in the proof of Proposition 6 in the companion paper. None of the programs associated with these points orders more than two process moves. From the assumption that P^I_{PMP} is full-dimensional, these points belong to $\mathcal{F}_s$. Since they are affinely independent, inequality (9) is facet-defining for P^I_{PMP} and for P^M_{PMP} by Proposition 4. □

In the next proposition, we state that the k-unicycle inequalities (and also inequalities (2) that indeed correspond to the 1-unicycle inequalities) define facets of P^M_{PMP}.

Proposition 9 *Let $I \subseteq M$ with $|I| = k$ and let $i_0 \in M \setminus I$. The k-unicycle inequality*

$$\delta_{i_0 i_0} + \sum_{i \in I} (\delta_{i i_0} + \delta_{i_0 i}) - \sum_{i \in I} \sum_{j \in I \setminus \{i\}} \delta_{ij} \leq 1 \tag{10}$$

is facet-defining for P^M_{PMP} .

Proof. Given $m, m' \in M$ with $m \neq m'$, we have

$$A_{mm'} = \begin{cases} \{[i_0, m']\} & \text{if } m = i_0,\ m' \in I, \\ \{[m, i_0]\} & \text{if } m \in I,\ m' = i_0, \\ \{[m, m'; i_0]\} & \text{if } m \in I,\ m' \in I, \\ \{[m, m']\} & \text{if } m \notin I \cup \{i_0\},\ m' \in I, \\ \{[m, m']\} & \text{if } m \in I,\ m' \notin I \cup \{i_0\}, \\ \{[m, m']\} & \text{if } m \notin I \cup \{i_0\},\ m' \notin I \cup \{i_0\}, \\ \{[i_0, m'; i] : i \in I\} & \text{if } m = i_0,\ m' \notin I, \\ \{[m, i_0; i] : i \in I\} & \text{if } m \notin I,\ m' = i_0. \end{cases}$$

We clearly have $M_s = I \cup \{i_0\}$, $F_1 = I$ and $F_2 = \{i_0\}$. The program consisting of interrupting all the process moves of M corresponds to a point of the face $\mathcal{F}$ of P^M_{PMP} induced by (10). Moreover from Proposition 1, the incomplete programs

$[m_0, i_0; i]$ and $[i_0, m_0; i]$ are admissible for any $m_0 \in M \setminus M_s$ and $i \in I$. It is then straightforward that these two incomplete programs induce points of $\mathcal{F}$. The two conditions of Proposition 3 are fulfilled and thus, we only need to focus on showing that inequality (10) is facet-defining for $P_{\text{PMP}}^{M_s}$. Let us consider the $(|I|+1)^2$ points introduced in the proof of Proposition 8 in the companion paper. The points of the first three sets correspond to programs ordering no more than two process moves. From the full-dimension assumption, they all belong to the face $\mathcal{F}_s$ of $P_{\text{PMP}}^{M_s}$ induced by inequality (10). The fourth set only contains points that correspond to programs $[m, m'; i_0]$ for all $m, m' \in M$ with $m \neq m'$ (i.e., $A_{mm'}$). From Lemma 1, all the points of this last set belong to $\mathcal{F}_s$. We then have found $(|I|+1)^2$ affinely-independent points of $\mathcal{F}_s$. Therefore, inequality (10) defines a facet of $P_{\text{PMP}}^{M_s}$, and by Proposition 3, of P_{PMP}^{M} as well. □

We now focus on the k-l-bicycle inequalities for which we give in the next proposition necessary and sufficient conditions to be facet-defining for P_{PMP}^{M}.

Proposition 10 *Let $i_0 \in M$, $j_0 \in M \setminus \{i_0\}$, $\emptyset \subset I \subset M \setminus \{i_0, j_0\}$ and $\emptyset \subset J \subset M \setminus \{i_0, j_0\}$ with $|I| = k$, $|J| = l$ and $I \cap J = \emptyset$. The k-l-bicycle inequality*

$$\delta_{i_0 i_0} + \delta_{j_0 j_0} + \delta_{i_0 j_0} + \sum_{i \in I} (\delta_{i i_0} + \delta_{i_0 i} - \delta_{i j_0}) + \sum_{j \in J} (\delta_{j j_0} + \delta_{j_0 j} - \delta_{i_0 j}) - \sum_{i \in I} \sum_{i' \in I \setminus \{i\}} \delta_{i i'} - \sum_{j \in J} \sum_{j' \in J \setminus \{j\}} \delta_{j j'} - \sum_{i \in I} \sum_{j \in J} \delta_{j i} \leq 2 \tag{11}$$

is facet-defining for P_{PMP}^{M} if and only if

i) $[j, i_0, j_0, i]$ is admissible for all $i \in I$ and $j \in J$,

ii) for all $i \in I$, there exists $j \in J$ so that $[i, j, i_0, j_0]$ is admissible,

iii) for all $j \in J$, there exists $i \in I$ so that $[i_0, j_0, i, j]$ is admissible,

iv) there exist $i \in I$ and $j \in I$ so that $[j_0, i, j, j_0]$.

Proof. We first prove the necessary conditions. For $i \in I$ and $m \in M \setminus \{i\}$, we have

$$A_{im} = \begin{cases} \{[i, i_0]\} & \text{if } m = i_0, \\ \{[i, j, i_0, j_0] : j \in J\} & \text{if } m = j_0, \\ \{[i, m; i_0]\} & \text{if } m \in I \setminus \{i\}, \\ \{[i, m]\} & \text{if } m \in J, \\ \{[i, m]\} & \text{if } m \notin I \cup J \cup \{i_0, j_0\}. \end{cases}$$

From the full-dimension assumption on P_{PMP}^{M} and Lemma 1, A_{im} contains an admissible program for all $m \in M \setminus \{j_0\}$. The set A_{ij_0} contains an admissible program if there exists $j \in J$ so that $[i, j, i_0, j_0]$ is admissible. The necessary

condition ii) is then proved. For $j \in J$ and $m \in M \setminus \{j\}$, we have

$$A_{jm} = \begin{cases} \{[j, j_0]\} & \text{if } m = j_0, \\ \{[j, i_0, j_0], \{[i, j, i_0] : i \in I\}\} & \text{if } m = i_0, \\ \{[j, m; j_0]\} & \text{if } m \in J \setminus \{j\}, \\ \{[j, i_0, j_0, m]\} & \text{if } m \in I, \\ \{[j, m]\} & \text{if } m \notin I \cup J \cup \{i_0, j_0\}. \end{cases}$$

As above, we then deduce the necessary condition i), that is, for all $j \in J$ and $i \in I$, $[j, i_0, j_0, i]$ is admissible. Remark that A_{ji_0} implies either that $[j, i_0, j_0]$ is admissible or that there exists $i \in I$ so that $[i, j, i_0]$ is admissible. The admissibility of $[j, i_0, j_0]$ is indeed implied by the one of $[j, i_0, j_0, i]$ for $i \in I$. Consider now the process moves i_0 and $m \in M \setminus \{i_0\}$. We have

$$A_{i_0m} = \begin{cases} \{\{[i_0, j_0, i] : i \in I\}, \{[j, i_0, j_0] : j \in J\}\} & \text{if } m = j_0, \\ \{[i_0, m]\} & \text{if } m \in I, \\ \{[i_0, j_0, i, m] : i \in I\} & \text{if } m \in J, \\ \{\{[i_0, m; i] : i \in I\}, \{[j, i_0, m, j_0] : j \in J\}\} & \text{if } m \notin I \cup J \cup \{i_0, j_0\}. \end{cases}$$

As above, we deduce that for all $j \in J$ there must exist $i \in I$ so that $[i_0, j_0, i, j]$ is admissible. Remark that the admissibility of the programs of $A_{i_0j_0}$ is obviously implied by Condition i). We then obtain necessary condition iii). Using similar arguments for j_0 and $m \in M \setminus \{j_0\}$ where

$$A_{j_0m} = \begin{cases} \{[j_0, i, j, i_0] : i \in I,\ j \in J\} & \text{if } m = i_0, \\ \{\{[j_0, m, j] : j \in J\}, \{[i_0, j_0, m\}\} & \text{if } m \in I, \\ \{[j_0, m]\} & \text{if } m \in J, \\ \{[j_0, m; j] : j \in J\} & \text{if } m \notin I \cup J \cup \{i_0, j_0\}, \end{cases}$$

we obtain necessary condition iv). Finally, let $m \notin I \cup J \cup \{i_0, j_0\}$ and $m' \in M \setminus \{m\}$. We have

$$A_{mm'} = \begin{cases} \{[m, i_0; i] : i \in I\} & \text{if } m' = i_0, \\ \{[m, j_0; j] : j \in J\} & \text{if } m' = j_0, \\ \{[m, m']\} & \text{if } m' \in I, \\ \{[m, m']\} & \text{if } m' \in J, \\ \{[m, m']\} & \text{if } m' \notin I \cup J \cup \{i_0, j_0, m\}. \end{cases}$$

From the full-dimension assumption on P^M_{PMP} and Lemma 1, $A_{mm'}$ contains an admissible program for all $m \notin I \cup J \cup \{i_0, j_0\}$ and $m' \in M$.

We now prove the sufficiency of the conditions. We clearly have $M_s = I \cup J \cup \{i_0, j_0\}$, $F_1 = I \cup J$ and $F_2 = \{i_0, j_0\}$. As we did in the proof of the previous propositions of this section, we will only focus on showing that inequality (11) is facet-defining for $P^{M_s}_{\text{PMP}}$. In fact, condition i) of Proposition 3 is trivially satisfied, and because of Lemma 1, the incomplete programs $[m_0, i_0; i]$, $[m_0, j_0; j]$,

$[i_0, m_0; i]$ and $[j_0, m_0; j]$ are admissible and correspond to points of the face of P^M_{PMP} induced by (11), for any $m_0 \in M \setminus M_s$, $i \in I$ and $j \in J$. Consider then the $|M_s|^2$ affinely-independent points introduced in the proof of Proposition 10 in the companion paper. These points are sorted into 15 different sets. The first five sets contain points associated with programs ordering at most two process moves. The assumed full dimension of $P^{M_s}_{\text{PMP}}$ implies that all these points belong to $\mathcal{F}_s = \{\delta \in P^{M_s}_{\text{PMP}} : (11) \text{ is tight for } \delta\}$. The admissibility of the programs associated with the points in the sets 9 and 10 (these points correspond to incomplete programs of the form $[m, m'; m'']$) comes directly from Lemma 1, whereas conditions i)-iv) imply the admissibility of the programs associated with the points in the remaining sets. These $|M_s|^2$ points then belong to $\mathcal{F}_s$ and thus, inequality (11) defines a facet of $P^{M_s}_{\text{PMP}}$. From Proposition 3, it also defines a facet of P^M_{PMP}. □

As we proved in the companion paper, the separation problems for inequalities (9), (10) and (11) are NP-hard. The proofs are based on showing that an algorithm solving any of these separation problems would also solve the max-cut problem.

3.2 Cover inequalities

We now turn our attention to capacity-related facet-defining inequalities for P^M_{PMP}. The underlying idea of these inequalities comes from the well-known cover inequalities (Balas, 1975 ; Hammer et al., 1975 ; Wolsey, 1975) which represent an important class of facets of the binary knapsack problem. For our problem, we can actually see a *cover* as a set of process moves that cannot be on a same processor at the same time.

Proposition 11 *Let $m_0 \in M$, $\emptyset \subset A \subseteq T(s_{m_0})$ and $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ be so that*

$$\sum_{m \in A} w_m > K_{s_{m_0}} + \sum_{m \in \overline{B}} w_m \tag{12}$$

with $\overline{B} = S(s_{m_0}) \setminus (B \cup \{m_0\})$. The source cover inequality

$$\sum_{m \in A} \delta_{m m_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}) \tag{13}$$

is valid for P^M_{PMP}.

Proof. Inequality (13) expresses the fact that all of the process moves in A cannot be performed (by migrations) if none of the process moves in $B \cup \{m_0\}$ have been performed (by migrations or interuptions), or in other words, that performing all the process moves in $\overline{B}$ does not free enough resources to perform all the ones in A.

Let $\overline{\delta}$ be a point of P^M_{PMP}. Suppose first that m_0 is interrupted, that is, $\overline{\delta}_{m_0m_0} = 1$ and then $\overline{\delta}_{mm_0} = \overline{\delta}_{m_0m} = 0$ for all $m \in M \setminus \{m_0\}$. Inequality (13) can then be rewritten $0 \leq 0$. Assume now that m_0 is not interrupted, that is, $\overline{\delta}_{m_0m_0} = 0$. If the left-hand side of (13) is equal to $|A| + |B|$, the program associated with $\overline{\delta}$ then migrates all the process moves in A before m_0 and all the process moves in B are performed after m_0. Remark that $|A| + |B|$ is the maximum possible value for the left-hand side of (13). Therefore, the process moves in $A \cup B \cup \{m_0\}$ are all together on processor s_{m_0}, and that implies

$$\sum_{m \in A} w_m \leq K_{s_{m_0}} + \sum_{m \in \overline{B}} w_m,$$

contradicting (12). The claim of validity follows. □

In the same manner, we can introduce the following inequalities by considering this time the target processor of a process move $m_0 \in M$ instead of its source processor as in Proposition 11.

Proposition 12 *Let $m_0 \in M$, $A \subseteq T(t_{m_0}) \setminus \{m_0\}$ and $\emptyset \subset B \subseteq S(t_{m_0})$ be so that*

$$w_{m_0} + \sum_{m \in A} w_m > K_{t_{m_0}} + \sum_{m \in \overline{B}} w_m \tag{14}$$

with $\overline{B} = S(t_{m_0}) \setminus B$. The target cover inequality

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0m} \leq (|A| + |B| - 1)(1 - \delta_{m_0m_0}) \tag{15}$$

is valid for P^M_{PMP}.

Proof. Condition (14) expresses the fact that all of the process moves in $A \cup \{m_0\}$ cannot be performed (by migrations) if none of the process moves in B have been performed (by interruptions or migrations), or in other words, that performing all the process moves in $\overline{B}$ does not free enough resources to perform all the ones in $A \cup \{m_0\}$.

The claim of validity follows from an argument similar to the end of the proof of Proposition 11. □

In order to clearly give necessary and sufficient conditions for the cover inequalities (13) and (15) to be facet-defining for P^M_{PMP}, we introduce some additional notation. Given $X \subseteq M$, let $m^X_{\min} = \text{argmin}\{w_m : m \in X\}$ (respectively $m^X_{\max} = \text{argmax}\{w_m : m \in X\}$) be a process move of X consuming the minimum (respectively maximum) amount of resource, that is, $w_m \geq w_{m^X_{\min}}$ (respectively

$w_m \leq w_{m^X_{\max}}$) for all $m \in X$. Given $u \in U$, $A \subseteq T(u)$ and $B \subseteq S(u)$, let $\lambda_{A,B} = w(A) - K_u - w(\overline{B})$ with $\overline{B} = S(u) \setminus B$.

Proposition 13 *A source cover inequality* (13) *defines a facet of* P^M_{PMP} *if and only if the following conditions hold*

i) $w_{m^A_{\min}} \geq \lambda_{A,B\cup\{m_0\}}$,
ii) $w_{m^B_{\min}} \geq \lambda_{A,B\cup\{m_0\}}$,
iii) if there exists $a \in A$ *so that* $B \cup \{m_0\} \subseteq T(s_a)$, *then either*
  a) $w_{m_0} \geq \lambda_{A,B\cup\{m_0\}}$, *or*
  b) $w_{m_0} + w_{m^B_{\min}} \leq K_{s_a} + w(S(s_a)) - w_a$,
iv) $w_m \leq \max\{w_{m^A_{\max}}, w_{m^B_{\max}}\} - \lambda_{A,B\cup\{m_0\}}$ *for all* $m \in (T(s_{m_0}) \cup S(s_{m_0})) \setminus (A \cup B \cup \{m_0\})$.

Proof. We have $M_s = A \cup B \cup \{m_0\}$. We will first prove that inequality (13) defines a facet of $P^{M_s}_{\rm PMP}$ if and only if conditions i)-iii) hold. We will then use Proposition 3 to prove the necessary and sufficient conditions for (13) to be facet-defining for $P^M_{\rm PMP}$. We point out that throughout this proof, K_m, $S(s_m)$ and $T(s_m)$, for any $m \in M_s$, are considered with respect to the process move programming problem associated with M_s, that is, all the process moves of $M \setminus M_s$ are supposed interrupted. Let $\mathcal{F}_s$ be the face of $P^{M_s}_{\rm PMP}$ induced by a source cover inequality (13). We first prove the necessity of conditions i)-iii). Suppose $\mathcal{F}_s$ defines a facet of $P^{M_s}_{\rm PMP}$. Given $\overline{a} \in A$, there exists $\delta^1 \in \mathcal{F}_s$ so that $\delta^1_{\overline{a}m_0} = 0$ and $\delta^1_{m_0m_0} = 0$. In fact, if such a point did not exist, we would have $\mathcal{F}_s \subsetneq \{\delta \in P^{M_s}_{\rm PMP} : \delta_{m_0m_0} + \delta_{\overline{a}m_0} = 1\} \neq P^{M_s}_{\rm PMP}$ and $\mathcal{F}_s$ would not define a facet. Since $\delta^1_{m_0m_0} = 0$ and $\delta^1 \in \mathcal{F}_s$, we have

$$\sum_{a\in A} \delta^1_{am_0} + \sum_{b\in B} \delta^1_{m_0b} = |A| + |B| - 1.$$

Because of $\delta^1_{\overline{a}m_0} = 0$, we then get $\delta^1_{am_0} = 1$ for all $a \in A \setminus \{\overline{a}\}$ and $\delta^1_{m_0b} = 1$ for all $b \in B$. Therefore, we have $w(A \setminus \{\overline{a}\}) \leq K_{s_{m_0}} + w(\overline{B})$, that is, $w_{\overline{a}} \geq w(A) - (K_{s_{m_0}} + w(\overline{B})) = \lambda_{A,B\cup\{m_0\}}$. Condition i) then directly follows. Using similar arguments, we can prove the necessity of Condition ii). Consider now a process move $\overline{a} \in A$ so that $B \cup \{m_0\} \subseteq T(s_{\overline{a}})$. Since $\mathcal{F}_s$ defines a facet of $P^{M_s}_{\rm PMP}$, there exists $\delta^3 \in \mathcal{F}_s$ so that $\delta^3_{m_0\overline{a}} = 1$. We then clearly have $\delta^3_{m_0m_0} = 0$, $\delta^3_{am_0} = 1$ for all $a \in A \setminus \{\overline{a}\}$ and $\delta^3_{m_0b} = 1$ for all $b \in B$. Remark that right before migrating m_0, all the process moves in $M_s \setminus \{\overline{a}\}$ are together on processor s_{m_0}. Since $\lambda_{A,B\cup\{m_0\}} > 0$, there exists $B' \subseteq B$ so that $\delta^3_{b\overline{a}} = 1$ for all $b \in B'$. We then have $w(A) \leq K_{s_{m_0}} + w(\overline{B}) + w_{m_0} + w(B')$, that is, $\lambda_{A,B\cup\{m_0\}} - w(B') \leq w_{m_0}$. If $w_{m_0} \geq \lambda_{A,B\cup\{m_0\}}$, the point δ^3 can be chosen so that $B' = \emptyset$. We remind that the full-dimension assumption on $P^{M_s}_{\rm PMP}$ implies that the program $[m_0, \overline{a}]$ is admissible. Suppose now that $w_{m_0} < \lambda_{A,B\cup\{m_0\}}$. Since $B \cup \{m_0\} \subseteq T(s_{\overline{a}})$, we then must have $w_{m_0} + w(B') \leq K_{s_{\overline{a}}} + w(S(s_{\overline{a}})) - w_{\overline{a}}$. We thus directly obtain $w_{m_0} + w_{m^B_{\min}} \leq K_{s_{\overline{a}}} + w(S(s_{\overline{a}})) - w_{\overline{a}}$, and Condition iii) is proved.

We now prove the sufficiency of conditions i)-iii). Suppose they are satisfied. Let $X_1 = \{\delta^1, \ldots, \delta^{n_1}\}$ with $n_1 = n_s^2 - 2(n_s - 1)$ be the set of points so that

- $\delta^1_{mm} = 1$ for all $m \in M_s$, that is, all the process moves of M_s are interrupted,
- $\sum_{m \in M_s} \delta^i_{mm} = n_s - 1$ and $\delta^i_{m_0 m_0} = 1$ for $i = 2, \ldots, n_s$, that is, all the process moves of $M_s \setminus \{m_0\}$ but one are interrupted,
- $\sum_{m \in M} \delta^i_{mm} = n_s - 2$ and $\delta^i_{m_0 m_0} = 1$ for $i = n_s + 1, \ldots, n_1$, that is, all the process moves of $M_s \setminus \{m_0\}$ but two are interrupted.

The points δ^i for $i = 1, \ldots, n_s$ clearly belong to $\mathcal{F}_s$. Moreover since $P^{M_s}_{\text{PMP}}$ is assumed full-dimensional, we also have $\delta^i \in \mathcal{F}_s$ for $i = n_s + 1, \ldots, n_1$. Furthermore, it is straightforward to see that the points of X_1 are affinely independent. Consider now points for which $\delta_{m_0 m_0} = 0$, that is, points satisfying

$$\sum_{a \in A} \delta_{a m_0} + \sum_{b \in B} \delta_{m_0 b} = |A| + |B| - 1.$$

Let $A = \{a_1, \ldots, a_{|A|}\}$ and $B = \{b_1, \ldots, b_{|B|}\}$. Consider the points δ^{n_1+i} for $i = 1, \ldots, |A|$ so that

$$\sum_{a \in A} \delta^{n_1+i}_{a m_0} = |A| - 1 \;\text{ and }\; \delta^{n_1+i}_{a_i a_i} = 1.$$

Let $X_2 = X_1 \cup \{\delta^{n_1+1}, \ldots, \delta^{n_2}\}$ with $n_2 = n_1 + |A|$. By Condition i), the points δ^{n_1+i} for $i = 1, \ldots, |A|$ belong to $\mathcal{F}_s$. Moreover, they are affinely independent since point δ^{n_1+i}, $i = 1, \ldots, |A|$, is the only one for which $\delta_{m_0 m_0} + \delta_{a_i m_0} = 0$. Consider the points δ^{n_2+i} for $i = 1, \ldots, |B|$ so that

$$\sum_{b \in B} \delta^{n_2+i}_{m_0 b} = |B| - 1 \;\text{ and }\; \delta^{n_2+i}_{b_i b_i} = 1.$$

As above and using Condition ii), we set $X_3 = X_2 \cup \{\delta^{n_2+1}, \ldots, \delta^{n_3}\} \subseteq \mathcal{F}$ where $n_3 = n_2 + |B|$, and the points of X_3 are affinely independent. Let $X_4 = X_3 \cup \{\delta^{n_3+1}, \ldots, \delta^{n_4}\}$ with $n_4 = n_3 + |A|$ and the points δ^{n_3+i} for $i = 1, \ldots, |A|$ are so that

$$\sum_{a \in A} \delta^{n_3+i}_{a m_0} = |A| - 1 \;\text{ and }\; \delta^{n_3+i}_{m_0 a_i} = 1.$$

(Remark that all the process moves of B are migrated after m_0). If $w_{m_0} \geq \lambda_{A, B \cup \{m_0\}}$, the points δ^{n_3+i} for $i = 1, \ldots, |A|$ can be chosen so that

$$\sum_{b \in B} \delta^{n_3+i}_{a_i b} = |B|,$$

and they all belong to $\mathcal{F}_s$. If $w_{m_0} < \lambda_{A, B \cup \{m_0\}}$, the points δ^{n_3+i} for $i = 1, \ldots, |A|$ can be chosen so that

$$\sum_{b \in B \setminus \{m^B_{\min}\}} \delta^{n_3+i}_{a_i b} = |B| - 1 \;\text{ and }\; \delta^{n_3+i}_{m^B_{\min} a_i} = 1.$$

Given $i \in \{1, \ldots, |A|\}$, if $\{m_0, m^B_{\min}\} \subseteq T(s_{a_i})$ conditions ii) and iii.b) then imply that $\delta^{n_3+i} \in \mathcal{F}_s$, and if $\{m_0, m^B_{\min}\} \not\subseteq T(s_{a_i})$ then the full-dimension assumption on $P^{M_s}_{\text{PMP}}$ and Condition ii) imply that $\delta^{n_3+i} \in \mathcal{F}_s$. Therefore $X_4 \subseteq \mathcal{F}_s$ and since a point δ^{n_3+i}, $i = 1, \ldots, |A|$, is the only one of X_4 so that $\delta_{m_0 a_i} = 1$, the points of X_4 clearly are affinely independent. Let $X_5 = X_4 \cup \{\delta^{n_4+1}, \ldots, \delta^{n_5}\}$ with $n_5 = n_4 + |B|$ and the points δ^{n_4+i} for $i = 1, \ldots, |B|$ are so that

$$\sum_{b \in B} \delta^{n_4+i}_{m_0 b} = |B| - 1, \quad \delta^{n_4+i}_{b_i m_0} = 1 \quad \text{and} \quad \sum_{a \in A} \delta^{n_3+i}_{a b_i} = |A| - 1.$$

Remark that all the process moves of A are migrated before m_0. From conditions i)-ii) and the full-dimension assumption on $P^{M_s}_{\text{PMP}}$, we clearly have $X_5 \subseteq \mathcal{F}_s$. Since the point δ^{n_4+i}, $i = 1, \ldots, |B|$, is the only one having $\delta_{b_i m_0} = 1$ the points of X_5 are affinely independent. We then have exhibited $n_5 = n_s^2 - 2(n_s - 1) + 2(|A| + |B|) = n_s^2$ affinely independent points of $\mathcal{F}_s$. Recall that $n_s = |A| + |B| + 1$. Since $\mathcal{F}_s \subsetneq P^{M_s}_{\text{PMP}}$, inequality (13) defines a facet of $P^{M_s}_{\text{PMP}}$.

We now just need to show that by adding Condition iv), we obtain necessary and sufficient conditions for inequality (13) to be facet-defining for P^M_{PMP}. Assume that inequality (13) defines a facet $\mathcal{F}$ of P^M_{PMP}. Consider first a process move $x \in M_0 \cap T(s_{m_0})$. There must exist $\delta^t \in \mathcal{F}$ such that $\delta^t_{x m_0} = 1$. We then clearly have $\delta^t_{m_0 m_0} = 0$ and

$$\sum_{a \in A} \delta^t_{a m_0} + \sum_{b \in B} \delta^t_{m_0 b} = |A| + |B| - 1.$$

For some $m_s \in A \cup B$, we then must have $w_m + w(A) \leq K_{s_{m_0}} + w(\overline{B}) + w_{m_s}$, that is,

$$\begin{aligned} w_m &\leq w_{m_s} - (w(A) - K_{s_{m_0}} - w(\overline{B})) \\ &= w_{m_s} - \lambda_{A, B \cup \{m_0\}} \\ &\leq \max\{w_{m^A_{\max}}, w_{m^B_{\max}}\} - \lambda_{A, B \cup \{m_0\}}. \end{aligned}$$

We thus obtain Condition iv) for a process move of $M_0 \cap T(s_{m_0})$. By considering $x \in M_0 \cap S(s_{m_0})$ and a point of $\mathcal{F}$ such that $\delta_{m_0 x} = 1$, we can obtain Condition iii) for a process move of $M_0 \cap S(s_{m_0})$ as well. Therefore, Condition iv) is necessary. We will use Proposition 3 to prove the sufficiency. We clearly have $F_1 = A \cup B$ and $F_{n_s - 1} = \{m_0\}$. Remark that $F_h = \emptyset$ for all $h = 2, \ldots, n_s - 2$, and $F_{n_s} = \emptyset$. We only need to check if for any $x \in M_0$, there exist $X_a \subseteq F_1$ and $X_b \subseteq F_1$ so that the incomplete programs $[x, m_0; X_a]$ and $[m_0, x; X_b]$ are admissible and induce points that belong to $\mathcal{F}' = \{\delta \in P^M_{\text{PMP}} : (13) \text{ is tight for } \delta\}$. In fact, the first condition of Proposition 3 is straightforwardly satisfied. Let $x \in M_0$ and suppose $x \in T(s_{m_0})$. Using similar ideas, we can prove the claim if $x \in S(s_{m_0})$ or $x \notin T(s_{m_0}) \cup S(s_{m_0})$. Remark that some of the associated programs do not

need Condition iv) to be admissible. Denote by m_s a process move of $A \cup B$ so that $w_{m_s} = \max\{w_{m^A_{\max}}, w_{m^B_{\max}}\}$. From Condition iv), we have

$$\begin{aligned} w(A) + w_x &\leq w(A) + w_{m_s} - \lambda_{A,B\cup\{m_0\}} \\ &= w_{m_s} + K_{s_{m_0}} + w(\overline{B})) \end{aligned}$$

Set $X_a = X_b = (A \cup B) \setminus \{m_s\}$. Using the previous inequality, it can be shown that the incomplete program $[x, m_0; X_a]$ that consists of migrating all the process moves of $A \setminus \{m_s\}$, then migrating x, then migrating m_0, and finally migrating all the process moves of $B \setminus \{m_s\}$ is admissible and its associated point belongs to $\mathcal{F}'$. Similarly, by only switching the order of migrations of x and m_0 (i.e., m_0 is migrated before x this time) we have an admissible program $[m_0, x; X_b]$ inducing a point of $\mathcal{F}'$. Therefore by Proposition 3, conditions i)-iv) are sufficient for inequality (13) to be facet-defining for P^M_{PMP}. Our proof is then complete. □

In the same way, we can give necessary and sufficient conditions for the target cover inequality (15) to be facet-defining for P^M_{PMP}.

Proposition 14 *A target cover inequality* (15) *defines a facet of* P^M_{PMP} *if and only if the following conditions hold:*

i) $w_{m^A_{\min}} \geq \lambda_{A\cup\{m_0\},B}$,
ii) $w_{m^B_{\min}} \geq \lambda_{A\cup\{m_0\},B}$,
iii) if $(A \cup \{m_0\}) \subseteq S(s_{m_0})$ *and there exists* $b \in B$ *so that* $t_b = s_{m_0}$ *either*
  a) $w_{m_0} \geq \lambda_{A\cup\{m_0\},B}$, *or*
  b) $w_{m_0} + w_{m^A_{\min}} \leq K_{s_{m_0}} + w(S(s_{m_0})) - w_b$,
iv) $w_m \leq \max\{w_{m^A_{\max}}, w_{m^B_{\max}}\} - \lambda_{A\cup\{m_0\},B}$ *for all* $m \in (T(t_{m_0}) \cup S(t_{m_0})) \setminus (A \cup B \cup \{m_0\})$. □

We now turn our attention to the separation problems of inequalities (13) and (15). We actually show that separating either the source or target cover inequalities can be reduced to solving n knapsack problems.

Proposition 15 *The source cover inequalities* (13) *can be separated in pseudo-polynomial time.*

Proof. Let $m_0 \in M$ and $\delta^\star \in \mathbb{R}^{n^2}$. The separation problem for (13) asks for two sets $A \subseteq T(s_{m_0})$ and $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ that satisfy Condition (12) and

$$\sum_{m\in A} \delta^\star_{mm_0} + \sum_{m\in B} \delta^\star_{m_0 m} > (|A| + |B| - 1)(1 - \delta^\star_{m_0 m_0}), \tag{16}$$

if such two sets exist. For $m \in T(s_{m_0}) \cup S(s_{m_0}) \setminus \{m_0\}$, let $x_m = 1$ if and only if either $m \in A$ or $m \in B$. Since $w(\overline{B}) = w(S(s_{m_0})) - w(B) - w_{m_0}$, Condition (12)

can be rewritten

$$\sum_{m\in T(s_{m_0})} w_m x_m + \sum_{m\in S(s_{m_0})\setminus\{m_0\}} w_m x_m \geq K_{s_{m_0}} + \sum_{m\in S(s_{m_0})} w_m - w_{m_0} + 1$$

Since $|A| = \sum_{m\in T(s_{m_0})} x_m$ and $|B| = \sum_{m\in S(s_{m_0})\setminus\{m_0\}} x_m$, (16) can be rewritten (after some rearrangements)

$$\sum_{m\in T(s_{m_0})} \alpha_m x_m + \sum_{m\in S(s_{m_0})\setminus\{m_0\}} \beta_m x_m < 1 - \delta^\star_{m_0 m_0}$$

where $\alpha_m = 1 - \delta^\star_{m m_0} - \delta^\star_{m_0 m_0}$ for all $m \in T(s_{m_0})$ and $\beta_m = 1 - \delta^\star_{m_0 m} - \delta^\star_{m_0 m_0}$ for all $m \in S(s_{m_0}) \setminus \{m_0\}$. Letting $y_m = 1 - x_m$ for all $m \in T(s_{m_0}) \cup S(s_{m_0}) \setminus \{m_0\}$ leads to the following knapsack problem

$$\begin{cases} z = \text{Maximize} \displaystyle\sum_{m\in T(s_{m_0})} \alpha_m y_m + \sum_{m\in S(s_{m_0})\setminus\{m_0\}} \beta_m y_m \\ \text{s. t.} \\ \displaystyle\sum_{m\in T(s_{m_0})} w_m y_m + \sum_{m\in S(s_{m_0})\setminus\{m_0\}} w_m y_m \leq \sum_{m\in T(s_{m_0})} w_m - K_{s_{m_0}} - 1, \\ y_m \in \{0,1\} \qquad \text{for all } m \in T(s_{m_0}) \cup S(s_{m_0}) \setminus \{m_0\}. \end{cases}$$

It is obvious that this knapsack problem has a feasible solution only if $w(T(s_{m_0}) - K_{s_{m_0}} - 1 \geq 0$, that is, there is not enough remaining capacity on processor s_{m_0} to migrate all the process moves of $T(s_{m_0})$ without performing a process move of $S(s_{m_0})$. If the optimal value $z^\star$ of this knapsack problem exists and

$$z^\star > \delta^\star_{m_0 m_0} - 1 + \sum_{m\in T(s_{m_0})} \alpha_m + \sum_{m\in S(s_{m_0})\setminus\{m_0\}} \beta_m,$$

a violated source cover inequality (13) generated by m_0 has then been found. Otherwise, it can be concluded that none exists.

Separating over the entire set of source cover inequalities can (13) therefore be performed by solving n knapsack problems such as the above. Each of this knapsack problem can be solved in pseudo-polynomial time using for instance the well-known Bellman recursion (Kellerer et al., 2004). The claim follows. □

Proposition 16 *The target cover inequalities* (15) *can be separated in pseudo-polynomial time.*

Proof. We proceed as in the proof of Proposition 15. The only difference lies in the considered knapsack problem. En fact, given $m_0 \in M$ and $\delta^\star \in \mathbb{R}^{n^2}$,

the separation problem for (15) turns out to reduce to the following knapsack problem.

$$\begin{cases} z = \text{Maximize} \displaystyle\sum_{m \in T(t_{m_0}) \setminus \{m_0\}} \alpha_m y_m + \sum_{m \in S(t_{m_0})} \beta_m y_m \\ \text{s. t.} \\ \displaystyle\sum_{m \in T(t_{m_0}) \setminus \{m_0\}} w_m y_m + \sum_{m \in S(t_{m_0})} w_m y_m \leq \sum_{m \in T(t_{m_0})} w_m - K_{t_{m_0}} - 1, \\ y_m \in \{0,1\} \qquad \text{for all } m \in (T(t_{m_0}) \setminus \{m_0\}) \cup S(t_{m_0}), \end{cases}$$

where $\alpha_m = 1 - \delta^\star_{mm_0} - \delta^\star_{m_0 m_0}$ for all $m \in T(s_{m_0}) \setminus \{m_0\}$ and $\beta_m = 1 - \delta^\star_{m_0 m} - \delta^\star_{m_0 m_0}$ for all $m \in S(s_{m_0})$. □

3.3 A cover-inequalities-based formulation

The cover inequalities previously introduced enable us to give an integer linear programming formulation of the PMP problem based only on facet-defining inequalities for P^M_{PMP}.

Lemma 2 *A vector δ of $\mathbb{R}^{n^2}$ is the characteristic vector associated with a solution of the process move programming problem if and only if it belongs to the set defined by the following constraints*

- *2-clique inequalities (i.e., $\delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1$ for all $\{m, m'\} \subseteq M$),*
- *1-unicycle inequalities (i.e., $\delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1$ for $m \neq m' \in M$),*
- *extended transitivity inequalities (i.e., $\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} + \delta_{m'm'} \leq 1$ for $m \neq m' \neq m'' \in M$),*
- *source cover inequalities* (13),
- *target cover inequalities* (15),
- *$\delta_{mm'} \in \{0,1\}$ for $m,\ m' \in M$.*

Proof. From the results obtained in this paper and its companion one, it is obvious that any solution to the PMP problem has its characteristic vector that satisfies all these constraints.

Let $\overline{\delta}$ be a solution of this system of constraints. If this solution does not induce a solution to the PMP problem, there then exists a process moves $\overline{m}$ so that the available capacity on its target processor $t_{\overline{m}}$ is not sufficient when it is migrated. Let A (respectively B) be the set of process moves having $t_{\overline{m}}$ as a target (respectively source) processor and that have already been migrated to $t_{\overline{m}}$ (respectively migrated from $t_{\overline{m}}$ or interrupted). We then have

$$w(A) + w_{\overline{m}} > K_{t_{\overline{m}}} + w(\overline{B})$$

where $\overline{B} = S(s_{t_{\overline{m}}}) \setminus B$. Therefore the target cover inequalities induced by $\overline{m}$, A and B is violated by $\overline{\delta}$, a contradiction. □

4 Conclusion

In this paper, we have studied the polytope P^M_{PMP} associated with the solution of the process move programming problem. We first have obtained necessary and sufficient conditions for the facet-defining inequalities introduced in the companion paper (Sirdey & Kerivin, 2006c) for the partial linear ordering polytope to be facet-defining for P^M_{PMP}. We then have introduced capacity-related facet-defining inequalities for P^M_{PMP} that can be separated in pseudo-polynomial time, and strengthen the integer linear programming formulation of the PMP problem we started with. Lastly, Sirdey & Kerivin (2006a) have devised a branch-and-cut algorithm based on this strengthened formulation given in Section 3.3. Their extensive computational results show the practical relevance of the formulation in terms of both exact and approximate resolution when the instance size increases. It would be interesting to extend their branch-and-cut algorithm by also considering inequalities (9)-(11).

A branch-and-cut algorithm for a resource-constrained scheduling problem *†

Renaud Sirdey[a] [b] and Hervé Kerivin[c]

[a] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.
[b] UMR CNRS Heudiasyc (Université de Technologie de Compiègne), Centre de recherches de Royallieu, BP 20529, 60205 Compiègne Cedex, France.
[c] UMR CNRS Limos (Université de Clermont-Ferrand II), Complexe scientifique des Cézeaux, 63177 Aubière Cedex, France.

Submitted on March 16, 2006
Revised on May 23, 2006
Revised on July 18, 2006
Accepted on September 29, 2006

Abstract

This paper is devoted to the exact resolution of a strongly NP-hard resource-constrained scheduling problem, the *Process Move Programming problem*, which arises in relation to the operability of certain high-availability real-time distributed systems. Based on the study of the polytope defined as the convex hull of the incidence vectors of the admissible process move programs, we present a branch-and-cut algorithm along with extensive computational results demonstrating its practical relevance, in terms of both exact and approximate resolution when the instance size increases.

Keywords: Polyhedral combinatorics, scheduling, branch-and-cut, distributed systems.

Mathematics Subject Classification: 90C57, 68M14.

∗. This research was supported in part by Association Nationale de la Recherche Technique grant CIFRE-121/2004.

†. Technical report Nortel GSM Access R&D PE/BSC/INF/017912 V01/EN, appeared in RAIRO Operations Research 41:235-251, 2007.

1 Introduction

In this paper, we present a branch-and-cut algorithm for the *Process Move Programming* (PMP) problem. This problem arises in relation to the operability of certain high-availability distributed switching systems. For example (Sirdey et al., 2003), consider a telecom switch managing radio cells on a set of call processing modules, hereafter referred to as *processors*, of finite capacity in terms of erlangs, CPU, memory, ports, etc.; each radio cell being managed by a dedicated process running on some processor. During network operation, some cells may be dynamically added, modified (transreceivers may be added or removed) or removed, potentially leading to unsatisfactory resource utilisation in the system. This issue is addressed by first obtaining a better system configuration and by subsequently reconfiguring the system, without violation of the capacity constraints on the processors.

Figure 1 provides an example of an instance of the PMP problem for a system with 10 processors, one resource and 46 processes. The capacity of each of the processors is equal to 30 and the sum of the consumptions of the processes is 281. The top and bottom figures respectively represent the initial and the final system states. For example, process number 23 must be moved from processor 2 to processor 6.

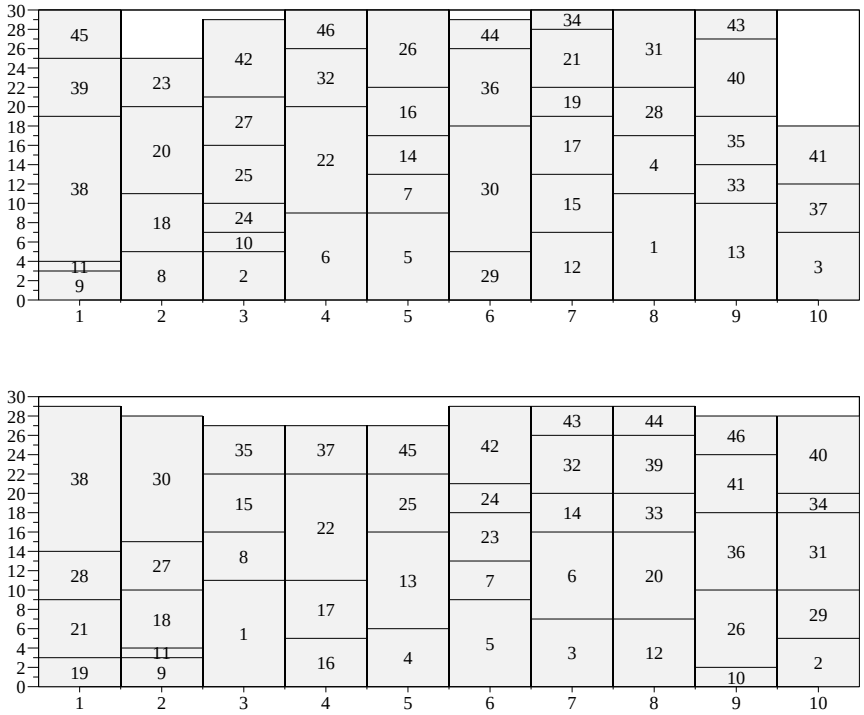


Figure 1: Example of an instance of the PMP problem.

We now proceed with a formal definition of the problem.

Let us consider a distributed system composed of a set U of *processors*, each processor offering an amount $C_u \in \mathbb{N}$ of a given resource. We are also given a set P of applications, hereafter referred to as *processes*, which consume the resources offered by the processors. The set P is sometimes referred to as the *payload* of the system. For each process $p \in P$, $w_p \in \mathbb{N}$ denotes the amount of resource which is consumed by process p. Note that neither C_u nor w_p vary with time.

An *admissible state* for the system is defined as a mapping $f : P \longrightarrow U \cup \{u_\infty\}$, where u_∞ is a dummy processor having infinite capacity, such that for all $u \in U$ we have

$$\sum_{p \in P(u;f)} w_p \leq C_u, \tag{1}$$

where $P(u; f) = \{p \in P : f(p) = u\}$. The processes in $\bar{P}(f) = P(u_\infty; f)$ are not instantiated and, when this set is non empty, the system is in *degraded mode*.

An instance of the PMP problem is then specified by two arbitrary system states f_i and f_t respectively referred to as the *initial system state* and the *final system state* or, for short, the *initial state* and the *final state*[1].

A process may be moved from one processor to another in two different ways: either it is *migrated*, in which case it consumes resources on both processors for the duration of the migration and this operation has virtually no impact on service, or it is *interrupted*, that is removed from the first processor and later restarted on the other one. Of course, this latter operation has an impact on service. Additionally, it is required that the capacity constraints (1) are always satisfied during the reconfiguration and that a process is moved (i.e., migrated or interrupted) at most once. This latest constraint is motivated by the fact that a process migration is far from being a lightweight operation (for reasons related to distributed data consistency which are out of the scope of this paper, see e.g. Jalote, 1994) and, as a consequence, it is desirable to avoid processes hopping around processors.

Throughout this paper, when it is said that a move is *interrupted*, it is meant that the process associated to the move is interrupted. This slightly abusive terminology significantly lightens our discourse.

For each processor u, a process p in $P(u; f_i) \setminus P(u; f_t)$ must be moved from u to $f_t(p)$. Let M denote the set of process moves thus induced by the initial and final states. Then for each $m \in M$, w_m, s_m and t_m respectively denote the amount of resource consumed by the process moved by m, the processor from which the process is moved that is the *source* of the move and the processor to which the process is moved that is the *target* of the move. Lastly, $S(u) = \{m \in M : s_m = u\}$

1. Throughout the rest of this paper, it is assumed that $\bar{P}(f_i) = \bar{P}(f_t) = \emptyset$. When this is not the case the processes in $\bar{P}(f_t) \setminus \bar{P}(f_i)$ should be stopped before the reconfiguration, hence some resources are freed, the processes in $\bar{P}(f_i) \setminus \bar{P}(f_t)$ should be started after the reconfiguration and the processes in $\bar{P}(f_i) \cap \bar{P}(f_t)$ are irrelevant.

and $T(u) = \{m \in M : t_m = u\}$.

A pair (I, σ), where $I \subseteq M$ and $\sigma : M \setminus I \longrightarrow \{1, \ldots, |M \setminus I|\}$ is a bijection, defines an admissible *process move program*, if provided that the moves in I are interrupted (for operational reasons, the interruptions are performed at the beginning) the other moves can be performed according to σ without inducing any violation of the capacity constraints (1). Formally, (I, σ) is an admissible program if for all $m \in M \setminus I$ we have

$$w_m \leq K_{t_m} + \sum_{\substack{m' \in I \\ s_{m'} = t_m}} w_{m'} + \sum_{\substack{m' \in S(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'} - \sum_{\substack{m' \in T(t_m) \setminus I \\ \sigma(m') < \sigma(m)}} w_{m'}, \tag{2}$$

where $K_u = C_u - \sum_{p \in P(u;f_i)} w_p$ denotes the remaining capacity on processor u in the initial state, thereby guaranteeing that the intermediate states are admissible.

Also note that because the final state is admissible, we have for each processor $u \in U$

$$K_u + \sum_{m \in S(u)} w_m - \sum_{m \in T(u)} w_m \geq 0. \tag{3}$$

Let c_m denote the cost of interrupting $m \in M$, the PMP problem then formally consists, given a set of moves, in finding a pair (I, σ) such that $c(I) = \sum_{m \in I} c_m$ is minimum.

Previous research on distributed system reconfiguration and other related problems (Coffman et al., 1983, 1985 ; Carlier, 1984a ; Hall et al., 2001 ; Saia, 2001 ; Anderson et al., 2001 ; Aggarwal et al., 2003) has mainly dealt with the approximate minimization of makespan related criteria under a set of constraints on the legal parallelism and has either ignored or only considered quite loose capacity constraints. As emphasized throughout this paper, the PMP problem or, more precisely, our model of the PMP problem shares some features with the linear ordering problem, which consists in finding a spanning acyclic tournament of maximum weight in a complete weighted digraph (Reinelt, 1985). In terms of exact resolution, Grötschel et al. (1985b) report solving real-world instances having up to 60 vertices with an algorithm that could be considered the first branch-and-cut ever. More recently, Christof & Reinelt (1998) were able to solve "hard" instances having up to around 80 vertices with a parallel branch-and-cut algorithm using facets from low-dimensional polytopes. Lastly, Mitchell & Borchers (2000) report solving instances with up to 250 vertices with a combined interior point/simplex cutting plane algorithm, though their instances appear easier than those of Christof and Reinelt. Needless to emphasize that, despite of certain structural similarities between the two problems, the presence of the capacity constraint implies that the PMP problem is in essence fairly different from the linear ordering problem.

Sirdey et al. (2007) have shown that the PMP problem is strongly NP-hard, exhibited some polynomially solvable special cases (the most notable one being

$|R| = 1$ and $w_m = w$ for all $m \in M$) as well as proposed a "combinatorial" branch-and-bound algorithm for the general case. Also, they provided an thorough literature survey. Additionally, approximate resolution algorithms have been proposed by Sirdey et al. (2009, 2010) (simulated annealing-based approach and Grasp-based approach, respectively). Lastly, in a series of two papers (Sirdey & Kerivin, 2006c ; Kerivin & Sirdey, 2006) we have investigated the PMP problem from the polyhedral viewpoint: we formulated the PMP problem as an integer linear program and introduced several classes of valid inequalities, all of which being facet-defining for the associated polytope under mildly restrictive assumptions.

This paper intends to assess the practical relevance of the latter polyhedral results. To the best of the authors' knowledge, this is the first application of the polyhedral approach to a distributed system reconfiguration problem as well as the first attempt to tackle such a problem exactly, to the exception of the branch-and-bound algorithm presented in Sirdey et al. (2007).

The paper is organized as follows. In Section 2, we provide the theoretical background laying at the basis of our algorithm. Our branch-and-cut algorithm is then presented in Section 3. Finally, we provide, in Section 4, extensive computational results which demonstrate the practical usefulness of the approach.

2 Polyhedral combinatorics of the PMP problem

In this section, we provide the theoretical background required in order to make this paper self-contained. We present an integer linear programming formulation of the problem along with a selection of facet-defining inequalities which are used in the algorithm. All the results in this section are proved in Sirdey & Kerivin (2006c) ; Kerivin & Sirdey (2006), proofs are therefore omitted.

2.1 An integer linear programming formulation

In this section, we formulate the PMP problem as an integer linear program. We first focus on obtaining a formulation for the decision problem which asks whether or not there exists an admissible process move program of the form $(\emptyset, \sigma)$, that is an admissible program of zero cost. We subsequently refine the model so as to encompass the notion of interruption.

For each ordered pair of distinct moves of M, say m and m', we introduce the *linear ordering variables* (Queyranne & Schulz, 1994)

$$\delta_{mm'} = \begin{cases} 1 & \text{if } m \text{ precedes } m', \\ 0 & \text{otherwise.} \end{cases}$$

In order for these variables to define a valid ordering, it is natural to ask for the following constraints to be satisfied

$$\begin{cases} \delta_{mm'} + \delta_{m'm} = 1 & \forall\{m, m'\} \subseteq M, \quad (4) \\ \delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M. \quad (5) \end{cases}$$

Constraints of type (4) simply express that either m precedes m' or m' precedes m. Constraints of type (5) are known as the *transitivity constraints* and simply state that if m precedes m' and if m' precedes m'' then m precedes m''. Along with the constraints

$$\delta_{mm'} \in \{0, 1\} \qquad m \neq m' \in M, \quad (6)$$

constraints of types (4) and (5) describe a *linear ordering polytope*, that is the convex hull of the incidence vectors of the linear orderings of the moves in M (see for example Grötschel et al., 1985a ; Fishburn, 1992 for details).

Since interruptions are (so far) disallowed, constraints of type (2) can be expressed as follows

$$w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} \delta_{m'm} - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} \qquad \forall m \in M. \quad (7)$$

It follows that any integral solution to the linear system of inequalities defined by the sets of constraints (4), (5), (6) and (7) (should such a solution exists) provides an admissible process move program of zero cost.

We now turn to the PMP problem and start by, for each move $m \in M$, introducing the variables

$$\delta_{mm} = \begin{cases} 1 & \text{if } m \text{ is interrupted,} \\ 0 & \text{otherwise.} \end{cases}$$

Constraints of type (2) can then be written as follows

$$(1 - \delta_{mm}) w_m \leq K_{t_m} + \sum_{m' \in S(t_m)} w_{m'} (\delta_{m'm'} + \delta_{m'm}) - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm}, \quad (8)$$

for all $m \in M$. The transitivity constraints (5) remain unchanged and constraints of type (4) must be replaced by constraints

$$\delta_{mm'} + \delta_{m'm} = 1 - \max(\delta_{mm}, \delta_{m'm'}) \qquad \forall\{m, m'\} \subseteq M. \quad (9)$$

These constraints simply express that if either m or m' is interrupted then neither m precedes m' nor m' precedes m (recall that the interruptions are performed at the beginning). Additionally, constraints of type (9) are equivalent to the following set of constraints

$$\begin{cases} \delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall\{m, m'\} \subseteq M, \quad (10) \\ \delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M. \quad (11) \end{cases}$$

Inequalities of types (10) and (11) are respectively referred to as the 2-*clique* and 1-*unicycle* inequalities (Sirdey & Kerivin, 2006c).

The resulting integer linear program for the process move programming problem is given in Figure 2. The polytope associated to this program is hereafter referred to as the *process move program polytope* or, for short, the *PMP polytope* and denoted P^M_{PMP}.

Note that P^M_{PMP} is fully dimensional under mildly restrictive assumptions (Kerivin & Sirdey, 2006) and that full dimensionality is hereafter assumed unless stated otherwise.

2.2 Facets of the process move program polytope

It turns out that the 2-clique inequalities (10) and the 1-unicycle inequalities (11) along with inequalities $\delta_{mm'} \geq 0$ for all $m, m' \in M$ with $m \neq m'$ define facets of P^M_{PMP}.

This is not the case for inequalities $\delta_{mm} \geq 0$ for all $m \in M$, for inequalities $\delta_{mm'} \leq 1$ for all $m, m' \in M$ as well as for the transitivity constraints (5) and the capacity constraints (8).

The transitivity constraints can however be replaced by the *extended transitivity constraints*

$$\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} + \delta_{m'm'} \leq 1 \qquad m \neq m' \neq m'' \neq m \in M, \tag{12}$$

which are facet-defining for P^M_{PMP}.

Let $m_0 \in M$ and let $\emptyset \subset A \subseteq T(s_{m_0})$ and $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ be such that

$$\sum_{m \in A} w_m > K_{s_{m_0}} + \sum_{m \in \bar{B}} w_m, \tag{13}$$

with $\bar{B} = S(s_{m_0}) \setminus (B \cup \{m_0\})$ and define the *source cover inequality* generated by m_0, A and B as

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}). \tag{14}$$

Also, let $m_0 \in M$ and let $A \subseteq T(t_{m_0}) \setminus \{m_0\}$ and $\emptyset \subset B \subseteq S(t_{m_0})$ be such that

$$w_{m_0} + \sum_{m \in A} w_m > K_{t_{m_0}} + \sum_{m \in \bar{B}} w_m, \tag{15}$$

with $\bar{B} = S(t_{m_0}) \setminus B$, the *target cover inequality* generated by m_0, A and B is then defined as

$$\sum_{m \in A} \delta_{mm_0} + \sum_{m \in B} \delta_{m_0 m} \leq (|A| + |B| - 1)(1 - \delta_{m_0 m_0}). \tag{16}$$

$$
\begin{cases}
\text{Minimize } \displaystyle\sum_{m \in M} c_m \delta_{mm} & \\
\text{s. t.} & \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} + \delta_{m'm'} \geq 1 & \forall \{m, m'\} \subseteq M, \\
\delta_{mm'} + \delta_{m'm} + \delta_{mm} \leq 1 & m \neq m' \in M, \\
\delta_{mm'} + \delta_{m'm''} - \delta_{mm''} \leq 1 & m \neq m' \neq m'' \neq m \in M, \\
(1 - \delta_{mm}) w_m \leq K_{t_m} + \displaystyle\sum_{m' \in S(t_m)} w_{m'} (\delta_{m'm'} + \delta_{m'm}) - \sum_{m' \in T(t_m) \setminus \{m\}} w_{m'} \delta_{m'm} & \forall m \in M, \\
\delta_{mm'} \in \{0, 1\} & m, m' \in M.
\end{cases}
$$

Figure 2: Formulation of the PMP problem as an integer linear program.

These inequalities have the following meaning. Condition (13) (respectively (15)) expresses the fact that all of the moves in A (respectively $A \cup \{m_0\}$) cannot be performed if none of the moves in $B \cup \{m_0\}$ (respectively B) have been or, in other words, that performing all the moves in $\bar{B}$ does not free enough resources to perform all the moves in A (respectively $A \cup \{m_0\}$) and inequality (14) (respectively (16)) prevents that from happening as soon as m_0 is not interrupted.

It has been shown in Kerivin & Sirdey (2006) that the source and target cover inequalities are both valid (unconditionally) and facet-defining (under reasonably restrictive assumptions) for P^M_{PMP}. It also turns out, as we shall later see, that both the source and target cover inequalities can be separated in pseudopolynomial time.

Other classes of facet-defining inequalities for P^M_{PMP} which are not presently used in our branch-and-cut algorithm can be found in Sirdey & Kerivin (2006c) ; Kerivin & Sirdey (2006), in particular, the 2-clique and 1-unicycle inequalities were generalized, yielding the k-clique ($k \geq 2$) and k-unicycle ($k \geq 1$) inequalities which are facet-defining for P^M_{PMP}.

3 A branch-and-cut algorithm

Our branch-and-cut algorithm is based on the linear relaxation which includes the trivial, 2-clique and 1-unicycle inequalities ($0 \leq \delta_{mm'} \leq 1$ for all $m, m' \in M$, (10) and (11), respectively) along with the extended transitivity and capacity constraints ((12) and (8), respectively) as well as the source and target cover inequalities ((14) and (16), respectively). Because all of these constraints are polynomial in number, apart from the latter two which can be separated in pseudopolynomial time, this linear relaxation can, in theory, be solved in pseudopolynomial time using the ellipsoid algorithm (Grötschel et al., 1988).

3.1 Solving the relaxation

In practice, the above linear relaxation is solved using a cutting-plane algorithm.

Separation-wise, the $O(|M|^2)$ 2-clique inequalities, the $O(|M|^2)$ 1-unicycle inequalities and the $O(|M|^3)$ extended transitivity inequalities are handled by brute-force.

The separation of the source and target cover inequalities, on the other hand, requires the resolution of $2|M|$ knapsack problems, each of these being solved in pseudopolynomial time using the well-known Bellman recursion (Kellerer et al., 2004).

Indeed, given $m_0 \in M$ and $\delta^\star \in \mathbb{R}^{|M|^2}$, the separation problem for the source cover inequalities asks for two sets $A \subseteq T(s_{m_0})$ and $B \subseteq S(s_{m_0}) \setminus \{m_0\}$ which

satisfy condition (13) and such that

$$\sum_{m\in A}\delta^\star_{mm_0}+\sum_{m\in B}\delta^\star_{m_0m}>(|A|+|B|-1)(1-\delta^\star_{m_0m_0}). \tag{17}$$

For $m\in T(s_{m_0})\cup S(s_{m_0})\setminus\{m_0\}$, let $x_m=1$ if and only if either $m\in A$ or $m\in B$. Since $\sum_{m\in\bar{B}}w_m=\sum_{m\in S(s_{m_0})}w_m-\sum_{m\in B}w_m-w_{m_0}$, condition (13) can be rewritten

$$\sum_{m\in T(s_{m_0})}w_mx_m+\sum_{m\in S(s_{m_0})\setminus\{m_0\}}w_mx_m\geq K_{s_{m_0}}+\sum_{m\in S(s_{m_0})}w_m-w_{m_0}+1.$$

Since $|A|=\sum_{m\in T(s_{m_0})}x_m$ and $|B|=\sum_{m\in S(s_{m_0})\setminus\{m_0\}}x_m$, inequality (17) can be rewritten (after rearrangement)

$$\sum_{m\in T(s_{m_0})}\xi_mx_m+\sum_{m\in S(s_{m_0})\setminus\{m_0\}}\zeta_mx_m<1-\delta^\star_{m_0m_0},$$

where $\xi_m=1-\delta^\star_{mm_0}-\delta^\star_{m_0m_0}$ and $\zeta_m=1-\delta^\star_{m_0m}-\delta^\star_{m_0m_0}$. Letting $y_m=1-x_m$ leads to the following knapsack problem

$$\begin{cases} z=\text{Maximize}\displaystyle\sum_{m\in T(s_{m_0})}\xi_my_m+\sum_{m\in S(s_{m_0})\setminus\{m_0\}}\zeta_my_m,\\ \text{s. t.}\\ \displaystyle\sum_{m\in T(s_{m_0})}w_my_m+\sum_{m\in S(s_{m_0})\setminus\{m_0\}}w_my_m\leq\sum_{m\in T(s_{m_0})}w_m-K_{s_{m_0}}-1,\\ y_m\in\{0,1\},\ m\in T(s_{m_0})\cup S(s_{m_0})\setminus\{m_0\}. \end{cases}$$

Then, if z exists (that is the case only when $\sum_{m\in T(s_{m_0})}w_m-K_{s_{m_0}}-1\geq 0$) and if

$$z>\delta^\star_{m_0m_0}-1+\sum_{m\in T(s_{m_0})}\xi_m+\sum_{m\in S(s_{m_0})\setminus\{m_0\}}\zeta_m,$$

a violated source cover inequality generated by m_0 has been found. Otherwise, it can be concluded that none exists.

A similar argument applies to the separation problem for the target cover inequalities.

At the beginning of the cutting plane algorithm, only the trivial inequalities and the capacity constraints are included. Then, at each iteration, at most the 100 most violated 2-clique inequalities, at most the 100 most violated 1-unicycle inequalities and at most the 100 most violated extended transitivity constraints are added along with, for each move, the most violated source and target cover inequalities, if any.

The augmented linear relaxation is then reoptimized and, in order to keep its size reasonably small, we remove all the inequalities which have a positive slack

at the current optimum to the exception of those initially present in the program, that is the trivial inequalities and the capacity constraints, although this might result in some inequalities being added and removed a few times. Note that this technique has already been used by Grötschel et al. (1984) on the linear ordering problem which, as already stressed, shares some features (including having constraints which are polynomial in numbers and which cannot practically be handled in their entirety) with the present problem.

3.2 Overview of the algorithm

First, a "good" initial incumbent is obtained using the simulated annealing algorithm of Sirdey et al. (2009), this algorithm being designed so as to produce (α, β)-acceptable solutions that is, given α and $\beta \in [0, 1]$, to produce, with probability at least α, solutions of value less than or equal to $\mathrm{OPT} + \beta(S - \mathrm{OPT})$, where OPT is the value of an optimal solution and $S = \sum_m c_m$ is the value of the worst possible one which consists in interrupting all the moves. Still, these are *not* theoretical guarantees but extensive computational experiments reported in Sirdey et al. (2009) strongly suggest that, in practice, the algorithm actually meets its design intent when $\alpha = 0.95$ and $\beta = 0.05$ (these values also being those used in the present study).

At the root node of the search tree, the algorithm starts by solving the initial linear relaxation using the cutting-plane algorithm presented in Section 3.1. Early termination occurs if the ceiling of the current relaxation value becomes equal to the initial incumbent, in which case the optimality of the latter is established. Unless the solution of the relaxation is integral, hence optimal, branching occurs. Note that the ceiling of the value of the solution of the initial relaxation then serves as a global lower bound, hereafter denoted GLB.

Then, at each subsequent node of the search tree, the linear relaxation (in which some variables have been fixed) is solved, starting from the linear program obtained, before branching, at the parent node (note that the constraints are added or removed only locally). Early termination of the cutting-plane algorithm occurs either if the linear program is proven infeasible or if the ceiling of the current relaxation value becomes equal to the value of the incumbent. The incumbent is then updated if the solution of the relaxation is integral and, otherwise, branching occurs.

The search tree is traversed using depth-first search and a fairly simple branching scheme: a variable whose value, say v, is closest to $\frac{1}{2}$ in the relaxation is selected and 0 (respectively 1) is chosen first for branching if $v \leq \frac{1}{2}$ (respectively $v > \frac{1}{2}$), branching on 1 (respectively 0) subsequently occurs only if the incumbent is still greater than the global lower bound.

4 Computational experiments

In this section, we report on computational experiments carried out so as to assess the practical relevance of our branch-and-cut algorithm. These experiments have been performed on a Sun Ultra 10 workstation with a 440 MHz Sparc microprocessor, 512 MB of memory and the Solaris 5.8 operating system. The linear programs have been solved using COIN-OR implementation of the simplex algorithm [2]. Lastly, a time limit of four hours was imposed.

4.1 Instance generation

Given U the set of processors and C the processor capacity, an instance is generated as follows.

First, the set of candidate processes is built by drawing consumptions uniformly in $\{1, \ldots, C\}$ until $\sum_{p \in P} w_p \geq C|U|$. The initial state, f_i, is then generated by randomly assigning the processes to the processors: the processor to which a process is assigned is drawn uniformly from the set of processors which remaining capacity is sufficient (note that not all processes necessarily end up assigned to a processor). The final state, f_t, is built in the very same way to the exception that only the processes which are assigned to a processor in the initial state are considered. An instance is considered valid only if all the processes assigned to a processor in the initial state are also assigned to a processor in the final state. Invalid instances are discarded and the construction process is repeated until a valid instance is obtained. The set of moves is then built as explained in the introduction.

It should be emphasized that the above scheme generates instances for which the capacity constraints are extremely tight, instances which can be expected to be hard and, in particular, significantly harder than those occurring in practice. Indeed, as far as the system to which this work is to be applied (Sirdey et al., 2003) is concerned, the capacity constraints are fairly loose due to the fact that some spare capacity is provisioned for fault tolerance purpose and that this spare capacity is spread among all the processors. Additionally, it should be stressed that the system carries at most 100 processes and that a preprocessing technique, based on the fact that the properties of a system state are invariant by a permutation of the processors, is used to decrease the number of moves by around 25% on average. This preprocessing addresses the operational need to keep the number of moves as low as possible by solving a minimum cost bipartite matching problem so as to find the permutation of the processors which minimizes that number, this is achieved by letting $|P(u; f_i) \setminus P(u'; f_t)|$ be the cost of pairing processors u and u'.

Considering this, it turned out that our algorithm was able to solve virtually

2. `www.coin-or.org`.

all practical instances to optimality within a few seconds and that, as a consequence, we had to consider more aggressive instance generation schemes, such as the above, in order to fairly evaluate its performances.

Lastly, we have supposed that $c_m = w_m$, which is quite natural for our application as it is reasonable to assume that the amount of service provided by a process is proportional to the amount of resources it consumes.

4.2 Computational results

In our experiments, $|U|$, the number of processors, was ranging from 15 to 100 (with a step size of 5) and C, the processor capacity, was set to 100. For each value of $|U|$ a set of 10 instances were generated. Hence, the algorithm was tried on 190 instances which sizes, in terms of number of moves, range from 17 to 184.

When the time limit was reached the algorithm output the best solution it found along with an upper bound on the optimality gap.

Given a solution of value z, distance to optimality was measured using the ratio

$$d(z) = \frac{z - \mathrm{OPT}}{S - \mathrm{OPT}},$$

where OPT (OPT, when unknown, being replaced by GLB) and $S = \sum_m c_m$ respectively denote the value of an optimal solution and of the worst possible one, which simply consists in interrupting all the moves. This is consistent with the definition of (α, β)-acceptability (section 3.2). Additionally, $1 - d(z)$ can be interpreted either as a *differential approximation ratio* (recall that differential approximation is concerned with how far the value of a solution is from the worst possible value, see Demange & Paschos, 1996) or as a *conventional approximation ratio* (Garey & Johnson, 1979) for the maximization problem complementary to the PMP problem which asks to maximize the sum of the costs of the moves which are *not* interrupted.

Table 1 illustrates the results obtained using our algorithm on a set of instances with around 45 moves (column "$|M|$") and 25 processors (column "$|U|$"). Columns "z_0" and "$d(z_0)$" respectively indicate the value of the initial incumbent and the distance between that value and the optimum one. Columns "GLB", "d(GLB)", "# it." and "# cont." respectively provide the value of the initial relaxation, the distance between that value and the optimum one along with the number of iterations of the cutting-plane algorithm required to solve the relaxation and the number of constraints in the linear program before branching. Columns "$z^\star$", "# nodes" and "CPU" respectively indicate the value of an optimum solution, the number of nodes explored by the algorithm and the total running time. The average instance size, the average distance between the initial incumbent value and the optimal one as well as the average distance between the

initial relaxation value and the optimal one are indicated at the bottom of the table.

$\|U\|$	$\|M\|$	z_0	$d(z_0)$	GLB	d(GLB)	# it.	# cont.	$z^\star$	# nodes	CPU
25	45	84	0.00%	78	0.33%	144	3667	84	5	113.88 s
25	38	173	3.40%	110	0.00%	53	2394	110	3	36.35 s
25	50	161	2.26%	110	0.24%	225	4352	115	13	330.35 s
25	37	218	3.51%	155	0.00%	48	2529	155	1	49.78 s
25	36	206	2.96%	153	0.00%	54	2263	153	4	31.28 s
25	41	68	0.00%	68	0.00%	$\geq$99	-	68	1	70.85 s
25	40	118	2.01%	77	0.22%	73	2657	81	15	52.79 s
25	55	74	1.23%	49	0.00%	325	4652	49	5	329.06 s
25	47	158	1.27%	135	0.00%	151	3700	135	6	130.71 s
25	42	128	3.30%	69	0.00%	43	2671	69	2	39.09 s
	43.1		1.99%		0.08%					

Table 1: Illustration of the results obtained using our algorithm on a set of 10 instances with $|U| = 25$.

Note that for the fourth instance an integral solution was obtained before branching and that for the sixth one the initial incumbent was proven optimal before completing the resolution of the initial relaxation.

Table 2 provides a summary of the results which were obtained using our branch-and-cut algorithm on the overall instance set. For each value of $|U|$, column "$\overline{|M|}$" provides the average number of moves of the instances in the set, columns "# inst." and "# solved" respectively indicate the number of instances which were generated and the number of instances which the algorithm was able to solve to optimality within the four hours time limit, additionally, columns "$\bar{d}(z_0)$", "$\bar{d}$(GLB)", "$\bar{d}(z_f)$" give upper bounds on, respectively, the average distance between the initial incumbent value and the optimal one, the average distance between the initial relaxation value and the optimal one as well as the average optimality gap. For example, the third row ($|U| = 25$) is a summary of Table 1.

In terms of exact resolution, our algorithm was quite successful up to $|U| = 45$ (i.e., with instances of size up to around 80 moves) in the sense that most instances were either solved to optimality or with an integrality gap of less than 2% within the allowed four hours.

As the instance size increased, along with $|U|$, fewer instances ended up being solved to optimality, within the four hours limit. Nevertheless, the algorithm is still practically relevant as it was able to find solutions with an optimality gap of less than 5% for all but 7 of the instances on which it was tried.

Overall, the biggest instance which was solved to optimality within the four hours time limit had size 119 ($|U| = 70$), and was solved in 2 h 11 m 57 s, and

$\|U\|$	$\overline{\|M\|}$	# inst.	# solved	$\bar{d}(z_0)$	$\bar{d}$(GLB)	$\bar{d}(z_f)$
15	23.2	10	10	1.92%	0.07%	0.00%
20	34.1	10	10	1.20%	0.33%	0.00%
25	43.1	10	10	1.99%	0.08%	0.00%
30	53.1	10	6	≤2.01%	≤1.05%	≤0.74%
35	61.8	10	7	≤2.38%	≤0.98%	≤0.86%
40	67.5	10	6	≤2.23%	≤0.48%	≤0.43%
45	76.5	10	6	≤2.50%	≤1.15%	≤1.02%
50	88.5	10	3	≤2.69%	≤1.80%	≤1.72%
55	93.6	10	4	≤3.22%	≤1.34%	≤1.27%
60	108.4	10	2	≤2.80%	≤2.52%	≤2.52%
75	117.7	10	0	≤3.69%	≤3.69%	≤3.69%
70	122.3	10	1	≤3.32%	≤3.02%	≤3.00%
75	131.3	10	0	≤4.08%	≤4.08%	≤4.08%
80	137.4	10	0	≤3.30%	≤3.30%	≤3.30%
85	148.5	10	0	≤3.67%	≤3.67%	≤3.67%
90	157.4	10	0	≤3.79%	≤3.79%	≤3.79%
95	169.2	10	0	≤4.77%	≤4.77%	≤4.77%
100	177.2	10	0	≤4.62%	≤4.62%	≤4.62%

Table 2: Summary of the results obtained using our branch-and-cut algorithm on the overall instance set.

the smallest instance which was not solved to optimality had size 49 ($|U| = 30$), though the optimality gap was less than or equal to 1.02%. Additionally, for only one instance, of size 172, the bound on the optimality gap was greater than 6% (actually 6.12%). Table 3 indicates for each range of the optimality gap, the size of the smallest instance which was not solved and the size of the biggest instance which was solved with a gap bound in that range.

Gap	$]0\%, 1\%]$	$]1\%, 2\%]$	$]2\%, 3\%]$	$]3\%, 4\%]$	$]4\%, 5\%]$	$]5\%, 6\%]$
Smallest	49	56	56	85	119	172
Biggest	101	139	155	180	184	184

Table 3: Sizes of the smallest instance which was not solved and of the biggest instance which was solved within a given optimality gap bound range.

Lastly, it should be emphasized that from $|U| = 75$ onward, the initial incumbent was rarely improved during the branch-and-cut phase. This latter phase, however, allowed to obtain a reasonable estimate of the optimality gap. This illustrates the relevance of hybridizing a carefully designed metaheuristic, such as the simulated annealing algorithm used to obtain the initial incumbent (Sirdey et al., 2009), and a polyhedral bound when tackling problems in the realm of bigger instances.

Empirically, the branch-and-cut algorithm presented in this paper appears complementary to the combinatorial algorithm presented in Sirdey et al. (2007). Although the latter algorithm turns out being faster when dealing with small instances, due to the comparatively low per node computational cost, as well as with instances having relatively homogeneous process weights, mainly due to the presence of dominance relations which are particularly efficient in that context, it suffers from the lack of a strong lower bound. As emphasized by the above results, our branch-and-cut algorithm does not suffer from such a drawback: the strength of the linear programming bound presented in this paper allows it to tackle, either exactly or within a few percents to optimality, instances which are out of the reach of the aforementioned combinatorial algorithm. Table 4 illustrates this on the set of 10 instances of Table 1 [3]. Indeed, although the branch-and-bound algorithm was able to find an optimum solution for 6 of the 10 instances, it was able to complete the optimality proof for only 3 of them, within a one hour time limit. Moreover, on these 3 instances, the calculation time was longer than that of the branch-and-cut algorithm.

3. To be fair, the branch-and-bound algorithm was also provided with the initial incumbent obtained with the simulated annealing algorithm discussed in Section 3.2.

$\|M\|$	45	38	50	37	36
BC	113.88 s	36.35 s	330.35 s	49.78 s	31.28 s
BB	$> 3600^*$ s	103.08 s	> 3600 s	> 3600 s	2779.19 s
$\|M\|$	41	40	55	47	42
BC	70.85 s	52.79 s	329.06 s	130.71 s	39.09 s
BB	$> 3600^*$ s	381.07 s	> 3600 s	> 3600 s	$> 3600^*$ s

Table 4: Experimental comparison between our branch-and-cut algorithm and the branch-and-bound algorithm presented in Sirdey et al. (2007) on the set of 10 instances of Table 1. A "$*$" indicates that the algorithm was able to find an optimum solution but not to complete the optimality proof.

5 Conclusion

In this paper, we have proposed a branch-and-cut algorithm for the Process Move Programming problem, a strongly NP-hard scheduling problem which consists, starting from an arbitrary initial process distribution on the processors of a distributed system, in finding the least disruptive sequence of operations (non-impacting process migrations or temporary process interruptions) at the end of which the system ends up in another predefined arbitrary state. The main constraint is that the capacity of the processors must not be exceeded during the reconfiguration. This problem has applications in the design of high-availability real-time distributed switching systems such as the one discussed by Sirdey et al. (2003).

The main ingredient of our branch-and-cut algorithm is a linear relaxation which is made up of exponentially many inequalities which are facet-defining for the PMP polytope. This relaxation, which can theoretically be solved in pseudopolynomial time, is solved, at each node of the search tree, using a cutting-plane algorithm.

From an industrial perspective, it can be considered that the PMP problem is solved by this algorithm as it is able to close virtually all practical instances within a few seconds. Additionally, we have reported on computational experiments illustrating the practical relevance of the algorithm when used to solve instances significantly harder than those occurring in practice, in terms both of size and tightness of the capacity constraints. Indeed the algorithm was able to solve instances with up to 119 moves (70 processors) within a four hours time limit. Although one should only expect to have good chances to solve instances of size up to around 80 moves within a four hours limit, our experiments still suggest that, when the instance size increases, the truncated version of the algorithm has fairly good approximate resolution capabilities as it was able to provably obtain solutions with an optimality gap of less than 5% for most instances of size up to around 180 moves, still within the four hours time limit.

Lastly, note that further research work will be carried out so as to assess

the practical relevance of the classes of facet-defining inequalities identified in Kerivin & Sirdey (2006) and which, for simplicity sake, are so far not used in our branch-and-cut algorithm.

Acknowledgements

The authors wish to thank the two anonymous referees for several suggestions that led to improvements in the paper.

Combinatorial optimization problems in wireless switch design * †

Renaud Sirdey[a]

[a] Service d'architecture BSC (PC 12A7), Nortel GSM Access R&D, Parc d'activités de Magny-Châteaufort, 78928 Yvelines Cedex 09, France.

Submitted on August 14th, 2006
Revised on September 5th, 2006
Accepted on September the 8th, 2006

Abstract

The purpose of this paper is to illustrate the diversity of combinatorial problems encountered in the design of wireless switching systems. This is done via a representative selection of examples of *real* problems along with their associated solution methods. It should be emphasized that all the solution methods presented in this paper are successfully operating in the field at the time of writing.

Keywords: Combinatorial optimization, OR in telecommunications.

Mathematics Subject Classification: 90C27, 90B18.

1 Introduction

The telecommunication industry is a major provider of challenging combinatorial problems to the operations research community, usually in the areas of networks design, dimensioning and operation. An additional and perhaps lesser known source of combinatorial problems can be found in the design of the machines which operate within the networks.

Loosely speaking, a telecom switch is a system offering limited amounts of interrelated resources (circuits, ports, links, computing modules, disk storage, CPU, memory, etc.) which should be efficiently used. In such a context, it is not surprising for optimization problems to crop up. Additionally, many constraints

∗. Technical report Nortel GSM Access R&D PE/BSC/INF/20290 V01/EN, appeared in *4OR* 5:319-333, 2007.

†. Part of this work has been presented at the *Journées Franciliennes de Recherche Opérationnelle*, June 2006.

such as on the execution time, on the size or on the maintainability of the software limit the spectrum of applicable solution methods, especially when dealing with NP-hard problems.

This paper illustrates the diversity of combinatorial problems encountered in the field of telecom switch design via a representative selection of *real* problems, real in the sense that all the problems presented in this paper have really occurred and that the solution methods discussed herein, which are essentially based on known algorithmic techniques, are successfully operating in the field at the time of writing. These problems have cropped up in the context of Nortel's GSM Base Station Subsystem (BSS), the subset of a GSM network mainly in charge of transmission and reception on the radio path, which is under the responsibility of the Base Transceiver Stations (BTS), as well as of radio resources and handovers management, under the responsibility of the Base Station Controllers (BSC). Loosely speaking, a BTS is an entity comprising radio transmission and reception devices (including the antennas) which can be seen as a kind of complex radio modem. The geographical areas which a BTS covers using different frequencies are referred to as radio cells. A BSC is essentially a small switch connected, on one side, to several BTS and, on the other side, to the core of the network known as the Network and Switching Subsystem (NSS). The BSC is in particular responsible for the management of the radio interface (radio channels allocation and release, handover management, etc.) through the remote command of the BTS and of the mobiles. The reader is referred to, e.g., Mouly & Pautet (1992) ; Lagrange et al. (2000) for more details regarding the GSM network architecture.

The paper is organized as follows. Section 2 describes our vision of the practice of operations research in the field of telecom switch design. Each of the sections 3 to 5 subsequently presents an application. The applications discussed in sections 3 and 4 concern the BSC while the application presented in Section 5 is related to the BTS. Section 3 discusses a multiobjective radio cell configuration problem which is reduced to an assignment problem and tackled using network flow techniques. Section 4 presents a problem related to the management of the interface between two switches which is equivalent to a variant of the bin-packing problem, the *extensible bin-packing* problem, and dealt with using efficient approximate solution algorithms having quite satisfactory performance guarantees. Lastly, Section 5 is devoted to a problem which consists in minimizing the electrical consumption of a BTS during a power outage under a service constraint, this problem being a special case of a low-dimensional nonlinear knapsack problem which can be solved in polynomial time using dynamic programming.

It should be emphasized that this paper contains very few original results. Our goal has been to illustrate how some algorithms and methods published by the academic community have been adapted to effectively solve real problems as well as the surprising number of seemingly unrelated fields which have inspired our solution methods. It is our genuine belief that the applications discussed in sections 3 to 5, although far from being exhaustive, are representative.

2 Some thoughts on the practice of OR

Operations research and more particularly combinatorial optimization is, as in many other fields, of the uttermost practical relevance in the context of telecom switch design.

As an introductory example, let us consider the theory of NP-hardness. As humorously illustrated in a famous cartoon by Garey & Johnson (1979), this theory allows to convince a customer (be it a validation team or a network operator) that some problems cannot, in practice, be solved exactly (though usually after a little bit of education as many people tend not to be familiar with the notion of computational complexity). That fact does not depend on the skills of the involved research and development teams.

Of course, the industry has a strong preference for problems which are tractable *in practice*[1]. Facing such a problem is the best scenario, not only an efficient solution procedure is likely to be found in a book or paper but, and perhaps more importantly, communication with the customer is eased. Indeed, as long as the constraints and objective function are agreed upon, criticism of the end results is almost pointless: best possible solutions are always provided, efficiently.

Unfortunately, industrialists do not have the privilege of choosing their problems and intractable (i.e., NP-hard) problems often crop up due either to the intrinsic nature of the environment in which the system is to be embedded or simply to the fact that early high-level product specifications rarely take into account considerations such as the computational complexity of problems to be encountered downstream (often because only a small portion of these problems are foreseeable during the early stages of a system design).

Telecom switches are subject to real-time constraints and generally have a distributed architecture made up of many modules with limited resources. When facing intractable problems, these constraints limit the spectrum of applicable solution procedures to efficient approximate ones. Of course, this does not mean that exact solution procedures are useless: their primary role is to assist in the empirical assessment of the quality of the solutions provided by approximate solution ones, when appropriate (mainly in the absence of satisfactory performance guarantees).

Another point which should not be eluded is the simplicity requirement imposed on the solution methods (Sirdey, 2005). Nowadays, the maintenance of a product is more often than not under the responsibility of individuals other than those who designed it. As a consequence, in order not to jeopardize transfers of ownership, it is fundamental to achieve an appropriate balance between, on the one hand, the performances of a given method versus, on the other hand, the complexity of its implementation.

1. I.e., excluding problems which are tractable only in theory that is, for example, excluding problems so far known to be polynomially solvable only using the ellipsoid algorithm (Grötschel et al., 1988 ; Schrijver, 2004).

3 Optimum radio cell configuration

This section deals with the problem of optimally configuring a radio cell by assigning predefined groups of radio channels, also known as Time Division Multiple Access (TDMA) frames, to transceivers. A transceiver being an elementary device which can emit or receive continuously on a single frequency (at a given time).

3.1 Preliminaries

Let T denote a set of groups of radio channels, hereafter referred to as *groups*, and X denote a set of transceivers, hereafter referred to as *TRX*. Each group can be assigned to at most one TRX and each TRX can be assigned at most one group. Additionally, the set X is supplemented by a "dummy" TRX, denoted d, which may be assigned an arbitrary number of groups. A group which is assigned to a dummy TRX is effectively unassigned and when at least one group is unassigned the cell is in *degraded mode*, as not all the configured service is provided.

Because of hardware constraints a group may not be assignable to all TRX. The set of TRX to which a group $t \in T$ can be assigned is denoted $X(t)$. Note that for all $t \in T$, d belongs to $X(t)$.

Additionally, a set S of services is given. Each groups $t \in T$ *demands* a set $S(t) \subseteq S$ of services and each TRX $x \in X$ *offers* a set of services denoted by $S(x) \subseteq S$. A group $t \in T$ may be assigned to a TRX $x \in X$ which does not support all of its service set, that is such that $S(t) \setminus S(x) \neq \emptyset$, when this happens the group, hence the cell, is also in degraded mode.

Lastly, each group is given a *criticality* $0, 1, 2, \ldots$ The value of this parameter may or may not be related to the group service set. For example, certain groups must be assigned to a TRX if the cell is to support any traffic, the criticality of this kind of groups is usually set to 0 (the most critical) and they must be assigned "at all cost".

Roughly speaking, the problem then consists in finding the "best" assignment of the groups to the TRX in terms of the objective function described in the next section.

3.2 Objective function

For each group $t \in T$ and each TRX $x \in X(t) \cup \{d\}$, the following variables are introduced

$$\delta_{tx} = \begin{cases} 1 & \text{if } t \text{ is to be assigned to } x, \\ 0 & \text{otherwise.} \end{cases}$$

We now consider a hierarchy of objective functions which captures the various facets of the problem.

First, criticality must be taken into account. The most critical groups must be assigned even if in doing so all the groups of lesser criticality cannot be assigned. This is achieved by minimizing the following objective function,

$$f_1(\delta) = \sum_{t \in T} W(C(t))\delta_{tx}, \tag{1}$$

where $C(t)$ denote the criticality of group $t \in T$ and, H denoting the largest criticality value, where weight function W is defined recursively by $W(H) = 1$ and, for $c = H - 1, \ldots, 0$,

$$W(c) = W(c+1)(|T(c+1)| + 1),$$

with $T(c) = \{t \in T : C(t) = c\}$.

Second, service must be taken into account. This is done via the minimization of the following objective function

$$f_2(\delta) = \sum_{t \in T} \sum_{x \in X(t) \setminus \{d\}} |S(t) \setminus S(x)|\delta_{tx}. \tag{2}$$

Note that the term $|S(t) \setminus S(x)|$ may be replaced by a weighted sum over the set $S(t) \setminus S(x)$, when services have different impacts.

The third objective allows to deal with TRX failures, additions or removals. For example, TRX may fail at any time and, upon TRX failure, the system is required to reconfigure the cell so as to ensure optimal operation with one less TRX. The new optimal assignment should then be reached via a sequence of group reassignments of lowest impact. Note that all calls carried by a group are lost upon reassignment of the group and that the group radio channels are unavailable during the reassignment. Let $R(t)$ denote the cost of reassigning group t (e.g., a function of its criticality, as for the first objective) and let

$$\gamma_{tx} = \begin{cases} 1 & \text{if } t \text{ is } \textit{presently} \text{ assigned to } x, \\ 0 & \text{otherwise.} \end{cases}$$

Then, $t \in T$ is reassigned if and only if $\sum_{x \in X(t) \setminus \{d\}} (1 - \gamma_{tx})\delta_{tx} = 1$. Hence, the lowest impact is achieved by minimizing

$$f_3(\delta) = \sum_{t \in T} \sum_{x \in X(t) \setminus \{d\}} R(t)(1 - \gamma_{tx})\delta_{tx}. \tag{3}$$

Lastly, service is again taken into account by minimizing the following objective function

$$f_4(\delta) = \sum_{t \in T} \sum_{x \in X(t) \setminus \{d\}} (H - C(t) + 1)|S(x) \setminus S(t)|\delta_{tx}. \tag{4}$$

This latter objective ensures that the more critical groups are given their best fitting TRX, if possible. As a consequence, the more critical groups are less likely to be reassigned (e.g., upon failure of another TRX).

Finally, functions (1), (2), (3) and (4) are aggregated into one linear objective function

$$f(\delta) = \sum_{i=1}^{4} \lambda_i f_i(\delta), \tag{5}$$

where the λ_i have been chosen so as to enforce a strict criteria hierarchy, i.e., so as to enforce that an arbitrary improvement in any of the criteria $j > i$ does not justify a degradation of criterion i.

3.3 Solution method

Since the objective function (5) is linear, it is easy to see that the problem can be modelled using a network flow and dealt with any algorithm solving the minimum cost flow problem (Ahuja et al., 1993).

Figure 1 provides an example of network for an instance with 3 groups and 4 TRX. The nodes s, t and d respectively denote the *source*, the *sink* and the *dummy* TRX. Only the capacity of the arcs are indicated (the cost of an arc which either links s to a vertex or links a vertex to t is zero, for the other arcs the cost is given as specified in the previous section). The existence of a solution is guaranteed by the fact that the capacity of the $\{d, t\}$ arc is equal to the number of groups. On this example, $X(t_1) = \{x_1, x_3, x_4, d\}$ and $X(t_2) = X(t_3) = \{x_2, d\}$ (recall that d always belongs to $X(t_i)$), as a consequence, either t_2 or t_3 will be assigned to d, i.e., not assigned. Given a minimum cost s-t-flow on the above network, the only arc $\{t_i, x_j\}$ or $\{t_i, d\}$ which carries one unit out of vertex t_i indicates the TRX, x_j or d, to which t_i is assigned.

Our implementation was based on the Successive Shortest Path algorithm (Ahuja et al., 1993 ; Korte & Vygen, 2000) using 64 bits integer arithmetic so as to cope with the large coefficients generally present in the objective function.

At the time of writing, this method successfully operates in the field. Also, it is important to stress that the method was implemented in less than 200 lines of code and successfully replaced the few thousands lines of code of an ad hoc algorithm implemented in an earlier version of the system which, on top of being hardly maintainable, was known not to operate satisfactorily.

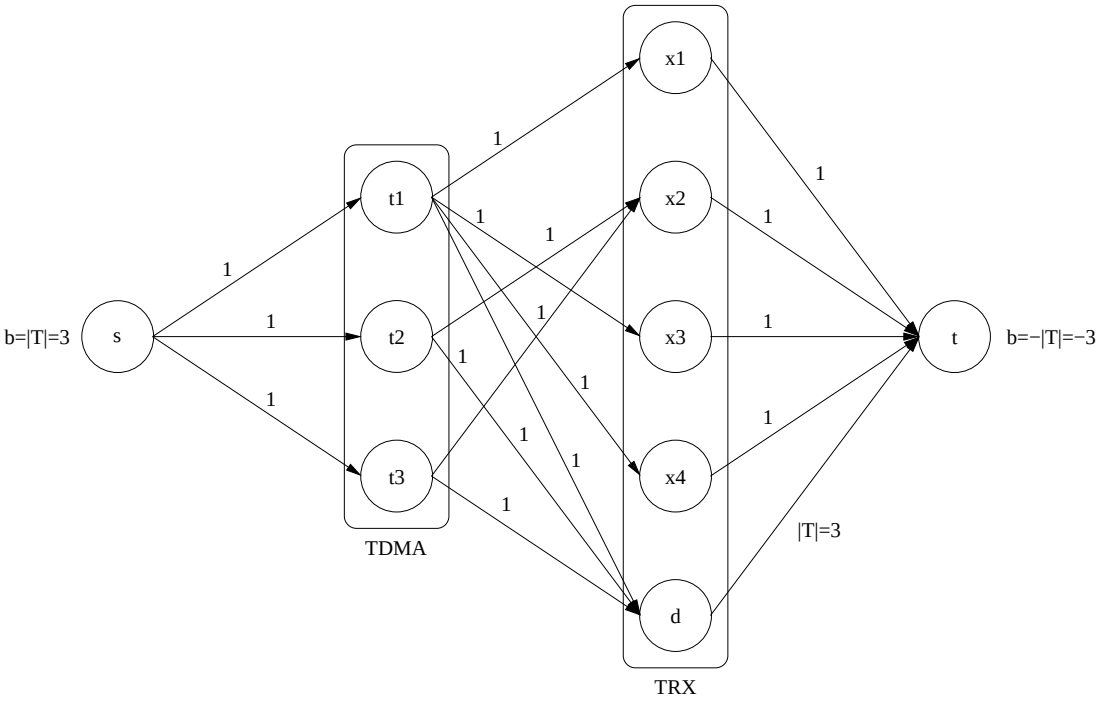


Figure 1: Example of network for an instance of the radio cell configuration problem of Section 3 with 3 groups and 4 TRX.

4 PCM interface management

This section is devoted to a problem related to the management of the PCM links[2] between two switches. The links offer some circuits which have to be allocated to a set of radio cells under the constraint that all the circuits allocated to a given cell must be on the same link (the cell is then said to be *assigned* to that link). Generally, be it due do the configuration itself or due to the unavailability of some of the links, the total circuit demand exceeds the total circuit capacity of the links and, hence, some cells will be allocated less circuits than required. The objective is thus to minimize the unsatisfied circuit demand.

4.1 Preliminaries

Let C denote a set of radio cells, hereafter referred to as *cells*, and let P denote a set of PCM links, hereafter referred to as *links*, available on the interface between two telecom switches. For a cell $c \in C$, let $w(c)$ denote the cell circuit requirement and for a link $p \in P$, let $K(p)$ denote the link circuit capacity.

2. A PCM (Pulse Code Modulation) link is a link carrying either 31 (European standard) or 24 (North-American standard) 64 kbit/s channels.

Generally, the total circuit demand exceeds the total circuit offer that is

$$\sum_{c \in C} w(c) > \sum_{p \in P} K(p),$$

and the problem consists in finding an assignment $f : C \longrightarrow P$ of the cells to the links so as to minimize the total overflow defined as

$$\sum_{p \in P} \max \left(0, \sum_{c \in C : f(c) = p} w(c) - K(p) \right).$$

It turns out that this problem is equivalent to the so-called *extensible bin-packing problem* (Dell'Olmo et al., 1998) which (paraphrasing the above), given a list of n positive numbers $\alpha_1, \ldots, \alpha_n$ and m bins $b_1, \ldots, b_m$ of capacities $k_1, \ldots, k_m$, asks for an assignment $f : \{1, \ldots, n\} \longrightarrow \{1, \ldots, m\}$ such that

$$\sum_{i=1}^{m} \max(k_i, \ell(b_i)),$$

where

$$\ell(b_i) = \sum_{j=1 \ldots n : f(j) = i} \alpha_j,$$

is minimum.

4.2 Solution method

Although the extensible bin-packing problem is NP-hard in the strong sense, there exist simple approximate solution algorithms with fairly reasonable performance guarantees.

Indeed, in the case where the bins have equal capacity Dell'Olmo et al. (1998) have established that the well-known LPT heuristic, which sorts the items in non-increasing order into a list and assigns, at each iteration, the first item of the list to a least loaded bin so far, is such that for any instance I of the problem

$$\frac{\mathrm{LPT}(I)}{\mathrm{OPT}(I)} \leq \frac{13}{12} \approx 1.0833. \tag{6}$$

Additionally, when the bins have different capacities, Dell'Olmo & Speranza (1999) have established that under the (reasonable) assumption that

$$\max_{i=1,\ldots,n} \alpha_i \leq \min_{i=1,\ldots,m} k_i, \tag{7}$$

the LPT heuristic (modified so as to consider the bin with the biggest idle space rather than the least load) is such that for any instance I of the problem [3]

$$\frac{\mathrm{LPT}(I)}{\mathrm{OPT}(I)} < 2(2-\sqrt{2}) \approx 1.1716. \tag{8}$$

The above motivates the following algorithm: sort the cells in order of decreasing circuit requirements $w_{c_1} \geq \ldots \geq w_{c_{|C|}}$, for $i = 1$ to $|C|$ if there exist a link $p \in P$ with positive remaining capacity, i.e., such that $K(p) > \sum_{j=1:f(c_j)=p}^{i-1} w_{c_j}$, then assign c_i to the link with the biggest remaining capacity otherwise assign c_i to the proportionally least loaded link, i.e., the link which minimizes

$$\frac{1}{K(p)} \sum_{j=1:f(c_j)=p}^{i-1} w_{c_j}.$$

It is clear that the above algorithm is equivalent to LPT when the links have the same capacity and, hence, that it provides ratio (6) as well as when the links have different capacities (something fairly rare, though possible, in our application) in which case it guarantees only ratio (8) as long as assumption (7) is satisfied, this assumption being more than reasonable for our application.

Although the algorithm which was finally implemented was slightly more complicated than the above, so as to cope with a few additional requirements which are not detailed here, the fact that such simple (and fast) heuristics provided acceptable performance guarantees was highly beneficial. In particular, it allowed to cut short some discussions with the validation team when coming with hand-generated examples of instances for which they were able to find solutions of better quality than those provided by the algorithm. Indeed, no trouble was found as long as the two solutions were within the tolerance derived from the performance guarantees and this allowed to elude a certain number of illegitimate enhancement requests, thereby avoiding adding undue complexity to the resulting software.

4.3 The parliament analogy

We now consider the problem of fairly allocating the circuits of a link to the cells which have been assigned to it, when demand exceeds offer.

Consider a country made up of a set of districts. Under the assumption that the country is democratic, it has a parliament with a small number of seats, small compared to overall country population.

A central problem in electoral systems theory is to find a fair repartition of the parliament seats between the districts, under some constraints such as that each district must be allocated at least one seat.

3. A tight bound of $\frac{8}{7} \approx 1.1429$ was conjectured in that paper.

It is generally admitted that a fair repartition algorithm must satisfy the following requirements (Balinski & Young, 2001):

1. If the number of seats increases then each district should be allocated at least the same number of seats.
2. If the population of a given district increases whereas the population of another decreases, then the latter should not be allocated more seats to the detriment of the earlier.
3. The repartition should be identical if one considers only a subset of the districts and the number of seats which have been allocated to them.
4. Perfect proportionality should be achieved when it is achievable.

It turns out that a very simple algorithm, known as the Sainte-Laguë method, fulfils all of the above requirements (Balinski & Young, 2001).

Returning to our per-link circuit allocation problem, the parliament analogy simply consists in seeing the set of cells as a set of districts, the circuit demand of a cell as a district population, the PCM as a parliament and the PCM circuit capacity as the number of seats in that parliament.

More generally, the analogy works with any set of consumers demanding certain amounts of a given scarce resource, when demand exceeds offer.

5 Battery life maximization

This section is devoted to a problem which consists in minimizing the electrical consumption of a Base Transceiver Station (BTS) during a power outage, under a service constraint. The BTS is made up of cabinets containing radio modules themselves containing TRX. Each TRX is assigned to one of the cells managed by the BTS. The service constraint is defined, for each cell, as the number of TRX which must remain operational, for as long as possible, during the power outage. As the electrical consumption of the site is proportional to the number of operational modules, the minimum consumption is achieved, in the monocabinet case, when the service constraint is satisfied with the smallest possible number of operational modules. The multicabinet case gives rise to a min-max problem: as each cabinet has its own battery, the time during which the service constraint is satisfied is limited by the lifespan of the battery of the cabinet hosting the biggest number of operational modules.

The purpose of such a functionality is to maximize the time during which a GSM network still covers a geographical area, though offering limited service, after a catastrophic power breakdown, e.g., after a natural disaster in which context the network can play a crucial role by still providing emergency services or even by helping locating trapped individuals when the wireline networks are damaged.

5.1 Preliminaries

Consider a BTS made up of a set K of cabinets as well as of a set R of radio modules, hereafter referred to as *modules*, each module containing up to 3 TRX. Also the BTS manages a set of cells, denoted by C.

Let W denote a $|R| \times |C|$ integer valued matrix which rows and columns are respectively indexed by the elements of R and C and such that given a module r and a cell c, W_{rc} denotes the number of TRX on r dedicated to cell c. Also, let L be a $|R| \times |K|$ binary valued matrix which rows and columns are respectively indexed by the element of R and K and such that given a module r and a cabinet k, $L_{rk} = 1$ if and only if module r is in cabinet k. Lastly, for each cell, a number N_c which specifies the TRX requirement during the power outage is given.

To each module is associated a binary variable x_r such that

$$x_r = \begin{cases} 1 & \text{if module } r \text{ is powered during the outage,} \\ 0 & \text{otherwise.} \end{cases}$$

The problem can then be modelled using the following nonlinear mathematical program

$$\begin{cases} \text{Minimize } \max\limits_{k \in K} \sum\limits_{r \in R} L_{rk} x_r, & (9) \\ \text{s. t.} \\ \sum\limits_{r \in R} W_{rc} x_r \geq N_c & \forall c \in C, \\ x_r \in \{0,1\} & \forall r \in R. \end{cases}$$

Assuming that $\sum_{r \in R} L_{rk} = L$ (i.e., that all the cabinets contain the same number of modules[4]), letting $x_r = 1 - y_r$ and rearranging leads to the following mathematical program

$$\begin{cases} \text{Maximize } \min\limits_{k \in K} \sum\limits_{r \in R} L_{rk} y_r, & (10) \\ \text{s. t.} \\ \sum\limits_{r \in R} W_{rc} y_r \leq \sum\limits_{r \in R} W_{rc} - N_c & \forall c \in C, \quad (11) \\ y_r \in \{0,1\} & \forall r \in R. \end{cases}$$

Stated as above, the problem is a Multidimensional Max-Min Knapsack Problem.

Given an optimal solution to the above mathematical program one has to switch off all modules for which $y_r = 1$ and for each cell to switch off $\sum_{r \in R} W_{rc} x_r - N_c$ TRX (no combination of TRX can improve the objective function, otherwise the solution would not be optimal).

4. When this is not true, dummy modules can be added. For such a modules, $W_{rc} = 0$ for each cell c.

5.2 The max-min knapsack problem

The Monodimensional Max-Min Knapsack Problem has been relatively recently introduced and studied by Yu (1996), see also Kellerer et al. (2004). This problem arises when one has to fill a knapsack with a selected set of items so that the minimum total profit gained under a set of scenarios is maximized, each item having a scenario-dependent profit (e.g., the profit of taking an umbrella for travel depends on the weather at destination). This problem has applications in the area of robust optimization under uncertainty, for example in robust capital budgeting.

The Monodimensional Max-Min Knapsack Problem can be stated as follows

$$\begin{cases} \text{Maximize } \min\limits_{k=1,\dots,t} \sum\limits_{j=1}^{n} p_{kj} x_j, & \\ \text{s. t.} & \\ \sum\limits_{j=1}^{n} w_j x_j \leq c, & \\ x_j \in \{0,1\} & j = 1,\dots,n, \end{cases} \tag{12}$$

When t is fixed, the problem is only weakly NP-hard and can be solved in pseudopolynomial time by dynamic programming. The solution method considers the following family of problems

$$\begin{cases} z_j(d; v_1,\dots,v_t) = \text{Maximize } \min\limits_{k=1,\dots,t} \sum\limits_{i=1}^{j} (p_{ki} x_i + v_k), & \\ \text{s. t.} & \\ \sum\limits_{i=1}^{j} w_i x_i \leq d, & \\ x_i \in \{0,1\} & i = 1,\dots,j, \end{cases} \tag{13}$$

where $z_n(c; 0,\dots,0)$ is the solution of problem (12). The solution method then uses the following recurrence formula

$$z_{j+1}(d; v_1,\dots,v_t) = \begin{cases} z_j(d; v_1,\dots,v_t) & \text{if } d < w_{j+1}, \\ \max\left\{ z_j(d; v_1,\dots,v_t), \right. & \\ \quad \left. z_j(d - w_{j+1}; v_1 + p_{1,j+1},\dots,v_t + p_{t,j+1}) \right\} & \text{otherwise.} \end{cases} \tag{14}$$

with

$$z_0(d; v_1,\dots,v_t) = \min_{k=1,\dots,t} v_k.$$

The resulting algorithm runs in $O(ncP^t)$, with $P = \max_{k=1,\dots,t} \sum_{j=1}^{n} p_{kj}$.

Problem (12) can easily be generalized to multiple dimensions yielding the Multidimensional Max-Min Knapsack Problem, that is the following mathematical program:

$$\begin{cases} \text{Maximize } \min\limits_{k=1,\ldots,t} \sum\limits_{j=1}^{n} p_{kj} x_j, & \\ \text{s. t.} & \\ \sum\limits_{j=1}^{n} w_{ij} x_j \leq c_i & i = 1, \ldots, m, \\ x_j \in \{0,1\} & j = 1, \ldots, n, \end{cases}$$

where all the inputs (p_{kj}, w_{ij} and c_i) are nonnegative integers.

5.3 Solution method

As in the case of the classical knapsack problem (Minoux, 1983 ; Kellerer et al., 2004), d (in equations (13) and (14) above) can be interpreted as a vector and the above formulas can then be used to derive a

$$O\left(n\left(\prod_{i=1}^{m} c_i\right) P^t\right)$$

pseudopolynomial algorithm solving the Multidimensional Max-Min Knapsack Problem, when both t and m are fixed.

However, provided that $n = |R|$, $c_i \leq O(|R|)$ and $0 \leq v_k \leq \sum_{j=1}^{n} p_{kj} \leq |R|$, the complexity of such an algorithm as far as our problem is concerned is

$$O\left(|R|^{|C|+|K|+1}\right),$$

hence polynomial for fixed $|C|$ and fixed $|K|$. This leads to a $O(|R|^7)$ algorithm for our application as 3 is an upper bound for both $|C|$ and $|K|$. Of course, this is not satisfactory in practice and rules out a direct dynamic programming implementation as such an implementation always realizes the worst case in terms of both time and space. Instead, we use a vector version of the above recurrence relation within a depth-first search algorithm relying on a data structure, referred to as the *cache*, which allows to filter out all of the search tree redundancy. The cache is keyed using integers uniquely associated to integer-valued vectors of the form $(j; d_1, \ldots, d_m; v_1, \ldots, v_t)$ ($0 \leq j \leq n$, $0 \leq d_i \leq c_i$ and $0 \leq v_k \leq \sum_{j=1}^{n} p_{kj}$). A cache item is defined as a pair

$$\{z_j(d_1, \ldots, d_m; v_1, \ldots, v_t), x_j\}$$

where x_j is a boolean variable which indicates whether or not item j belongs to the optimal solution of subproblem $z_j(d_1, \ldots, d_m; v_1, \ldots, v_t)$, this latter field

allowing to build the solution once the data structure has been filled. Thus, the subtree rooted at node $(j; d_1, \ldots, d_m; v_1, \ldots, v_t)$ of the search tree is explored if and only if no cache item is associated to that node, a cache item is then added after exploring the subtree. Also note that the cache is assumed to provide logarithmic access time, i.e., it is assumed to be some kind of balanced binary search tree (Knuth, 1998).

This algorithm may be considered a reinterpretation of a dynamic programming recurrence formula as a dominance relationship for use in a tree search algorithm. This point is further discussed by Ibaraki (1977).

In practice, it turns out that an implementation of this algorithm is able to solve the biggest practical instances (3 cabinets, 3 cells and 24 TRX per cell) in a bit more than 1 second on a standard laptop PC, a duration which is acceptable as power outages of less than a minute are filtered out. Table 1 illustrates the performances of the algorithm, note that for all instances $\sum_r W_{r1} = \sum_r W_{r2} = \sum_r W_{r3} = 24$.

N_1	N_2	N_3	# mod.	# TRX	CPU (s)	cache size
12	12	12	24	72	0.510	91 022
5	7	11	24	72	0.971	161 178
6	6	6	24	72	1.201	200 190
7	6	4	24	72	1.221	199 422
2	2	2	24	72	1.331	223 463

Table 1: Illustration of the performance of the depth-first search algorithm for solving the battery life maximization problem of Section 5 on a set of instances of biggest practical size.

6 Conclusion

In this paper, our intent has been to share some insights on the practical relevancy of using OR methods in the field of telecom switch design. In order to do so, we have provided a representative selection of *real* problems along with their associated solution methods:

- A polynomial radio cell configuration problem dealt with using network flow techniques.
- A strongly NP-hard radio-cell-to-link assignment problem tackled with simple approximation algorithms with acceptable performance guarantees (as well as a touch of electoral systems theory).
- A BTS battery life maximization problem which ends up being a polynomial special case of a nonlinear knapsack problem dealt with using a depth-first search algorithm supplemented with dominance relationships derived from a dynamic programming recurrence relation.

At the time of writing, all these solution methods are successfully operating in the field.

Although it is far from being exhaustive, other problems are discussed in Sirdey et al. (2003) and Sirdey et al. (2007), we believe that this list of problems is fairly representative of the kind of combinatorial problems encountered in telecom switch design. We also believe that the solution methods presented herein are representative of the spectrum of techniques which are usable under the constraints of the field (real-time, maintainability, etc.).

Lastly, we hope that we have been able to convincingly illustrate how research from academia can be almost directly and successfully applied to concrete telecom problems, even when the research comes from seemingly unrelated fields.

Conclusion et perspectives

On lit parfois, dans la littérature non spécialisée, que savoir qu'un problème est indécidable ou NP-difficile permet aux informaticiens de l'exclure d'emblée de leurs recherches. La réalité est bien différente [5] ! Nous venons de le voir.

Non content de nous permettre de résoudre toutes les instances réelles que nous leurs avons présentées, nos algorithmes nous ont aussi permis de résoudre de manière satisfaisante, c'est-à-dire au moins avec une distance à l'optimum prouvée être acceptable, presque toutes les instances difficiles, en termes à la fois de taille mais aussi de capacité résiduelle des processeurs, sur lesquelles nous les avons essayés.

En particulier, notre algorithme de recuit simulé est un candidat tout à fait viable, tant sur les plans de la qualité des solutions produites et du temps d'exécution que sur les plans de la complexité et de la maintenabilité du logiciel résultant, pour le remplacement de l'algorithme, quelque peu naïf [6], actuellement implémenté dans le système. Ceci d'autant que la tendance est au durcissement des contraintes de capacité, ce qui, nous l'avons vu, a une incidence directe sur la difficulté des instances, en raison des exigences qui pèsent sur l'accroissement de la capacité du système (afin de diminuer le coût unitaire par erlang) mais aussi de l'utilisation de la procédure de reconfiguration que nous, équipe BSC, envisageons dans un contexte d'amélioration des mécanismes de mise à jour logiciel sans arrêt du système.

En complément des algorithmes présentés dans la partie II de ce mémoire, nous avons travaillé sur une heuristique rapide, mais moins satisfaisante sur le plan théorique, basée sur Grasp [7], une métaheuristique, introduite dans les années 90 par Feo & Resende (1989, 1995), dont l'intérêt réside dans la combinaison des méthodes gloutonnes avec les méthodes de voisinage (dont nous avons parlées à la section 1.2.2). Il s'agit d'une méthode à démarrages multiples : à chaque étape un algorithme glouton randomisé est utilisé pour construire une solution réalisable dont le voisinage est exploré. La meilleure solution ainsi obtenue est conservée.

5. Pour les problèmes indécidables comme pour les problèmes NP-difficiles, nous avons en particulier à l'esprit la théorie de l'interprétation abstraite (Cousot & Cousot, 1977).

6. Spécifié par l'auteur avant le début de son travail de thèse. Il s'agit d'un algorithme glouton qui exploite néanmoins la méthode de décomposition basée sur la connexité forte présentée à la section 4.1.1.

7. *Greedy Randomized Adaptive Search Procedure.*

Nous avons introduit un algorithme glouton dont le principe repose sur l'insertion non invalidante de déplacements dans un ordonnancement réalisable. Cet algorithme possède un certain nombre de bonnes propriétés : à chaque étape tous les déplacements se voient offrir plusieurs opportunités d'insertion et toutes les interruptions sont remises en question. Une fois randomisé à la manière de Hart & Shogan (1987), cet algorithme nous donne le premier ingrédient de la méthode Grasp. Nous avons ensuite complété ce schéma par une méthode de voisinage simple, basée sur une stratégie d'échange entre les déplacements interrompus et ceux qui ne le sont pas : il s'agit simplement de s'assurer qu'interrompre un déplacement non interrompu ne permet pas d'insérer, sans l'invalider, un déplacement interrompu plus coûteux dans l'ordonnancement réalisable.

Malgré une distance moyenne à l'optimum satisfaisante sur notre base d'instances (1.68%), cet algorithme souffre de systématiquement manquer l'optimum sur certaines petites instances pathologiques (telles celle de la figure 2, page 138). En ce sens, bien qu'elle soit significativement moins coûteuse en temps de calcul que notre algorithme de recuit, la qualité des solutions obtenues à l'aide de cette approche s'avère moins stable.

Ces travaux font l'objet d'un article paru en 2010 dans un numéro spécial *Metaheuristics and Real-World Problems* du journal *International Journal of Innovative Computing and Applications* (volume 2, pages 143 à 149).

Aussi, il convient de souligner que le champs d'application de nos travaux peut être élargi à des problèmes de reroutage dans certains réseaux de télécommunications. En effet, le problème de reconfiguration se trouve être un cas particulier d'un problème de reconfiguration de chemins de routage dans les réseaux MPLS [8, 9] (Awduche, 1999). Grossièrement, il s'agit d'ordonner des déplacements de chemins de routage de telle manière que des contraintes de capacité sur les liens du réseau sous-jacent ne soient jamais violées, les situations de blocage étant résolues en supprimant temporairement des chemins (Józsa & Makai, 2003).

Il s'avère que les résultats et méthodes que nous avons introduits pour le problème de reconfiguration se généralisent tous assez naturellement à ce problème et qu'ils pourraient ainsi contribuer significativement à l'amélioration des méthodes de résolution disponibles. Nous avons déjà publié un rapport technique préliminaire sur ce sujet (Sirdey, 2006b).

Sur un plan plus général, enfin, nous espérons que les travaux présentés dans ce mémoire contribueront à montrer que les outils et méthodes issus de la recherche universitaire peuvent être appliqués avec succès à des problèmes industriels *concrets*. De même, nous espérons aussi avoir contribué à montrer que ces problèmes peuvent donner lieu à des problématiques de recherche particulièrement stimulantes, sans pour autant qu'ils soient vidés de leur substance.

8. *Multi-Protocol Label Switching*.

9. Nous souhaitons remercier Monsieur Oliver KLOPFENSTEIN, ingénieur de recherche chez France Télécom R&D, qui nous a suggéré cette équivalence.

Bibliographie

K. Aardal et S. van Hoesel. « Polyhedral techniques in combinatorial optimization I: theory ». *Statistica Neerlandica*, 50, 1996.

K. Aardal et S. van Hoesel. « Polyhedral techniques in combinatorial optimization II: applications and computations ». *Statistica Neerlandica*, 53, 1999.

E. H. L. Aarts et P. J. M. van Laarhoven. « Statistical cooling: a general approach to combinatorial optimization problems ». *Philips Journal of Research*, 40:193–226, 1985.

G. Aggarwal, R. Motwani et A. Zhu. « The load rebalancing problem ». Dans *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 258–265, 2003.

R. K. Ahuja, T. L. Magnanti et J. B. Orlin. *Network flows. Theory, algorithms and applications*. Prentice Hall, 1993.

E. Anderson, J. Hall, J. Hartline, M. Hobbs, A. R. Karlin, J. Saia, R. Swaminathan et J. Wilkes. « An experimental study of data migration algorithms ». Dans *Proceedings of the 5th International Workshop on Algorithm Engineering*, Lecture Notes in Computer Science, page 145. Springer, 2001.

D. Applegate, R. Bixby, V. Chvátal et W. Cook. « On the solution of traveling salesman problems ». *Documenta Mathematica*, Extra Volume ICM: 645–656, 1998.

Architecture BSC. « BSCe3 product specification ». Rapport technique PE/BSC/DD/0339 V02/EN, service d'architecture BSC, Nortel GSM Access R&D, mars 2003a.

Architecture BSC. « BSCe3/TCUe3 dimensioning ». Rapport technique PE/BSS/DD/3543 V02/EN, service d'architecture BSC, Nortel GSM Access R&D, janvier 2003b.

D. O. Awduche. « MPLS and traffic engineering in IP networks ». *IEEE Communication Magazine*, 37:42–47, 1999.

E. Balas. « Facets of the knapsack polytope ». *Mathematical Programming*, 8: 146–164, 1975.

M. L. Balinski et H. P. Young. *Fair representation: meeting the ideal of one man, one vote*. Brookings Institution Press, 2001.

J. Bang-Jensen et G. Gutin. *Digraphs—Theory, algorithms and applications*. Springer-Verlag, 2002.

P. Baptiste, J. Carlier et A. Jouglet. « A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates ». *European Journal of Operational Research*, 158:595–608, 2004.

R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

M. H. Bjorndal, A. Caprara, P. I. Cowling, F. Della Croce, H. Lourenço, F. Manucelli, A. J. Orman, D. Pisinger, C. Rego et J. J. Salazar. « Some thoughts on combinatorial optimization ». *European Journal of Operational Research*, 83:253–270, 1995.

P. Brucker. *Scheduling algorithms*. Springer, 2004.

P. Brucker, B. Jurisch et B. Sievers. « A branch-and-bound algorithm for the job-shop scheduling problem ». *Discrete Applied Mathematics*, 49:107–127, 1994.

P. Brucker et S. Knust. *Complex scheduling*. GOR-Publications. Springer, 2006.

J. Carlier. « Le problème de l'ordonnancement des paiements de dettes ». *RAIRO—Operations Research*, 18, février 1984a.

J. Carlier. *Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité*, tome 40 de *Méthodologie & Architecture des Systèmes Informatiques*. Université P. et M. Curie et CNRS, 1984b.

J. Carlier. « Communication personnelle », 2006.

J. Carlier et P. Chrétienne. *Problèmes d'ordonnancements : modélisation, complexité et algorithmes*. Études et Recherches en Informatique. Masson, 1988.

J. Carlier et E. Pinson. « A branch-and-bound method for solving the job shop problem ». *Management Science*, 35:164–176, 1989.

V. Cerny. « Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm ». *Journal of Optimization Theory and Applications*, 5:41–51, 1985.

T. Christof et G. Reinelt. « Algorithmic aspects of using small instance relaxations in parallel branch-and-cut ». Rapport technique, Université de Heidelberg, 1998.

E. G. Coffman, M. R. Garey, D. S. Johnson et A. S. Lapaugh. « Scheduling file transfers in distributed networks ». Dans *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 254–266, 1983.

E. G. Coffman, M. R. Garey, D. S. Johnson et A. S. Lapaugh. « Scheduling file transfers ». *SIAM Journal on Computing*, 14, 1985.

B. Colasse et F. Pavé. « La recherche opérationnelle entre acteurs et réalité, entretien avec B. Roy ». *Annales des Mines*, mars 1997.

S. A. Cook. « The complexity of theorem proving procedures ». Dans *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

P. Cousot et R. Cousot. « Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints ». Dans *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.

J.-P. Delahaye. « Un algorithme à un million de dollars ». *Pour La Science*, août 2005.

P. Dell'Olmo, H. Kellerer, M. G. Speranza et Z. Tuza. « A $\frac{13}{12}$ approximation algorithm for bin packing with extendable bins ». *Information Processing Letters*, 65:229–233, 1998.

P. Dell'Olmo et M. G. Speranza. « Approximation algorithms for partitioning small items in unequal bins to minimize the total size ». *Discrete Applied Mathematics*, 94:181–191, 1999.

K. Delvin. *The millenium problems. The seven greatest unsolved mathematical puzzles of our time.* Basic Books, 2002.

M. Demange et V. T. Paschos. « On an approximation measure founded on the links between optimization and polynomial approximation theory ». *Theoretical Computer Science*, 158:117–141, 1996.

F. Derepas. *Utilisation d'abstractions pour modéliser et vérifier des systèmes distribués utilisés en téléphonie cellulaire.* Thèse de doctorat, Université Denis Diderot, janvier 2002.

J. Dréo, A. Pétrowski, P. Siarry et É. Taillard. *Métaheuristiques pour l'optimisation difficile.* Algorithmes. Eyrolles, 2003.

T. A. Feo et M. G. C. Resende. « A probabilistic heuristic for a computationally difficult set covering problem ». *Operations Research Letters*, 8:67–71, 1989.

T. A. Feo et M. G. C. Resende. « Greedy randomized adaptive search procedure ». *Journal of Global Optimization*, 6:109–133, 1995.

S. Fiorini. *Polyhedral combinatorics of order polytopes*. Thèse de doctorat, Université Libre de Bruxelles, 2001.

P. C. Fishburn. « Induced binary probabilities and the linear ordering polytope: a status report ». *Mathematical Social Sciences*, 23:67–80, 1992.

M. Fowler. « Yet another optimization article ». *IEEE Software*, mai-juin 2004.

M. R. Garey et D. S. Johnson. *Computers and intractability—A guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

B. Gavish et O. R. Liu Sheng. « Dynamic file migration in distributed computer systems ». *Communications of the ACM*, 33:177–189, 1990.

S. B. Gelfand et S. K. Mitter. « Analysis of simulated annealing for optimisation ». Dans *Proceedings of the 24th IEEE Conference on Decision and Control*, pages 779–786, 1985.

R. L. Glass. « In search of meaning (a tale of two words) ». *IEEE Software*, pages 133–135, juillet-août 2002.

F. Glover, B. Alidaee, C. Rego et G. Kochenberger. « One-pass heuristics for large-scale unconstrained binary quadratic problems ». *European Journal of Operational Research*, 137:272–287, 2002.

R. E. Gomory. « Outline of an algorithm for integer solutions to linear programs ». *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

Gotha. « Les problèmes d'ordonnancements ». *RAIRO—Operations Research*, 27:77–150, 1993.

Gotha. *Modèles et algorithmes en ordonnancement*. Ellipses, 2004.

M. Grötschel, M. Jünger et G. Reinelt. « A cutting plane algorithm for the linear ordering problem ». *Operations Research*, 32:1195–1220, 1984.

M. Grötschel, M. Jünger et G. Reinelt. « Facets of the linear ordering polytope ». *Mathematical Programming*, 33:43–60, 1985a.

M. Grötschel, M. Jünger et G. Reinelt. « On the acyclic subgraph polytope ». *Mathematical Programming*, 33:28–42, 1985b.

M. Grötschel, L. Lovász et A. Schrijver. *Geometric algorithms and combinatorial optimization*, tome 2 de *Algorithms and Combinatorics*. Springer, 1988.

R. Guerraoui et A. Schiper. « Software-based replication for fault tolerance ». *Computer*, 30:68–74, 1997.

B. Hajek. « Cooling schedule for optimal annealing ». *Mathematics of Operations Research*, 13:311–329, 1988.

B. Hajek et G. Sasaki. « Simulated annealing – to cool or not ». *Systems & Control Letters*, 12:443–447, 1989.

J. Hall, J. Hartline, A. R. Karlin, J. Saia et J. Wilkes. « On algorithms for efficient data migration ». Dans *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 620–629, 2001.

P. L. Hammer, E. L. Johnson et U. N. Peled. « Facets of regular 0-1 polytopes ». *Mathematical Programming*, 8:179–206, 1975.

J. P. Hart et A. W. Shogan. « Semi-greedy heuristics: an empirical study ». *Operations Research Letters*, 6:107–114, 1987.

G. Hébuterne. *Écoulement du trafic dans les autocommutateurs*. Collection Technique et Scientifique des Télécommunications. Masson, 1985.

K. L. Hoffman et M. W. Padberg. « Solving airline crew scheduling problems by branch-and-cut ». *Management Science*, 39:657–682, 1993.

J. Hromkovič. *Algorithmics for hard problems. Introduction to combinatorial optimization, randomization, approximation, and heuristics*. Texts in Theoretical Computer Science. Springer, 2003.

J. Hudepohl, W. Snipes, J. Davies et R. Paterson. « Network element reliability, availability, maintainability and survivability requiremens for software and hardware ». Rapport technique TR-0S04-137-2002, Nortel Hardware and Software Dependability Design Group, 2004.

T. Ibaraki. « The power of dominance relations in branch-and-bound algorithms ». *Journal of the ACM*, 24:264–279, 1977.

P. Jalote. *Fault tolerance in distributed systems*. Distributed Systems. Prentice Hall, 1994.

M. Jerrum et G. B. Sorkin. « Simulated annealing for graph bisection ». Dans *Proceedings of the 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 94–103, 1993.

D. S. Johnson, C. R. Aragon, L. A. McGeogh et C. Schevon. « Optimization by simulated annealing—an experimental evaluation part I: graph partitioning ». *Operations Research*, 37:865–892, 1989.

D. S. Johnson, C. R. Aragon, L. A. McGeogh et C. Schevon. « Optimization by simulated annealing—an experimental evaluation part II: graph coloring and number partitioning ». *Operations Research*, 39:378–406, 1991.

A. Jouglet. *Ordonnancer une machine pour minimiser la somme des coûts*. Thèse de doctorat, Université de Technologie de Compiègne, 2002.

B. G. Józsa et M. Makai. « On the solution of reroute sequence planning problem in MPLS networks ». *Computer Networks*, 42:199–210, 2003.

R. M. Karp. « An introduction to randomized algorithms ». *Discrete Applied Mathematics*, 34:165–201, 1991.

H. Kellerer et U. Pferschy. « Improved dynamic programming in connection with an FPTAS for the knaspack problem ». *Journal of Combinatorial Optimization*, 3:59–71, 1999.

H. Kellerer, U. Pferschy et D. Pisinger. *Knapsack problems*. Springer, 2004.

J. G. Kemeny et J. L. Snell. *Finite Markov Chains*. The University Series in Undergraduate Mathematics. D. van Nostrand Company, Princeton, New Jersey, 1960.

H. Kerivin. *Réseaux fiables et polyèdres*. Thèse de doctorat, Université Blaise Pascal de Clermont-Ferrand, novembre 2000.

H. Kerivin et R. Sirdey. « Polyhedral combinatorics of a resource-constrained ordering problem part II: on the process move program polytope ». Rapport technique PE/BSC/INF/017913 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, France, 2006.

S. Kirkpatrick, C. D. Gelatt Jr et M. P. Vecchi. « Optimization by simulated annealing ». *Science*, mai 1983.

D. E. Knuth. « Structured programming with go to statements ». *Computing Surveys*, 6:261–301, 1974.

D. E. Knuth. *Sorting and searching*, tome 3 de *The Art of Computer Programming*. Addison-Wesley, 1998.

B. Korte et J. Vygen. *Combinatorial optimization—Theory and algorithms*, tome 21 de *Algorithms and Combinatorics*. Springer, 2000.

X. Lagrange, P. Godlewski et S. Tabbane. *Réseaux GSM, des principes à la norme*. Réseaux et télécommunications. Hermès Science Publications, 2000.

J. Lesourne. *Un homme de notre siècle. De Polytechnique à la prospective et au journal Le Monde*. Éditions Odile Jacob, 2000.

M. Li et P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Graduate Texts in Computer Science. Springer, 1997.

L. Lovász et M. D. Plummer. *Matching Theory*, tome 29 de *Annals of Discrete Mathematics*. North-Holland, 1986.

M. Lundy et A. Mees. « Convergence of an annealing algorithm ». *Mathematical Programming*, 34:111–124, 1986.

N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1996.

A. R. Mahjoub. « Approches polyédrales ». Dans V. Th. Paschos, éditeur, *Optimisation combinatoire: concepts fondamentaux*, pages 263–329. Hermes, 2005.

J.-F. Maurras. *Programmation linéaire. Complexité, séparation et optimisation*, tome 38 de *Mathématiques & Applications*. Springer et Société de Mathématiques Appliquées et Industrielles (SMAI), 2002.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller et E. Teller. « Equation of state calculations by fast computing machines ». *Journal of Chemical Physics*, 21:1087–1092, 1953.

M. Minoux. *Programmation mathématique. Théorie et algorithmes (tome 2)*. Collection Technique et Scientifique des Télécommunications. Dunod, 1983.

J. E. Mitchell et B. Borchers. « Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm ». Dans *High performance optimization*, pages 349–366. Kluwer, 2000.

A. Möbius, A. Neklioudov, A. Díaz-Sánchez, K. H. Hoffman, A. Fachat et M. Schreiber. « Optimization by thermal cycling ». *Physical Review Letters*, 79:4297–4301, 1997.

J. Monnot, V. T. Paschos et S. Toulouse. *Approximation polynomiale des problèmes NP-difficiles. Optima locaux et rapport différentiel*. Hermès Science Publications. Lavoisier, 2003.

M. Mouly et M.-B. Pautet. *The GSM System for Mobile Commnunications—A comprehensive overview of the European Digital Cellular Systems*. Telecom Publishing, 1992.

R. Müller. « On the poset polytope of a digraph ». *Mathematical Programming*, 73:31–49, 1996.

G. L. Nemhauser et L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1999.

A. Nolte et R. Schrader. « Simulated annealing and its problems to color graphs ». Dans *Algorithms—ESA 96*, tome 1136 de *Lecture Notes in Computer Science*, pages 138–151. Springer, 1996.

A. Pagès et M. Gondran. *Fiabilité des systèmes*, tome 39 de *Collection de la Direction des Études et Recherches d'Électricité de France*. Eyrolles, 1980.

D. Pisinger. « Where are the hard knapsack problems? ». *Computers & Operations Research*, 32:2271–2284, 2005.

M. Queyranne et A. S. Schulz. « Polyhedral approaches to machine scheduling ». Rapport technique 408/1994, Université de Technologie de Berlin, 1994.

M. Raynal. *La communication et le temps dans les réseaux et les systèmes répartis*, tome 75 de *Collection de la Direction des Études et Recherches d'Électricité de France*. Eyrolles, 1991.

M. Raynal. *Synchronisation et état global dans les systèmes répartis*, tome 79 de *Collection de la Direction des Études et Recherches d'Électricité de France*. Eyrolles, 1992.

G. Reinelt. *The linear ordering problem: algorithms and applications*, tome 8 de *Research and exposition in mathematics*. Heldermann Verlag Berlin, 1985.

S. Ross. *Stochastic Processes*. Probability and Statistics. John Wiley & Sons, 1996.

J. C. Saia. « Data migration with edge capacities and machine speeds ». Rapport technique, Université de Washington, 2001.

G. H. Sasaki et B. Hajek. « The time complexity of maximum matching by simulated annealing ». *Journal of the ACM*, 35:387–403, 1988.

A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1986.

A. Schrijver. *Combinatorial optimization—Polyhedra and efficiency*, tome 24 de *Algorithms and Combinatorics*. Springer, 2004.

M. Sipser. « The history and status of the P versus NP question ». Dans *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 603–618, 1992.

R. Sirdey. « A new algorithm for PCM defense on the A-Gprs interface ». Rapport technique PE/BSC/DD/010799 V02/EN, service d'architecture BSC, Nortel GSM Access R&D, 2004a.

R. Sirdey. « Reliability analysis for the BSCe3/TCUe3 ». Rapport technique PE/SYS/DD/11123 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, 2004b.

R. Sirdey. « BSC software engineering guidelines ». Rapport technique PE/BSC/APP/16486 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, 2005.

R. Sirdey. « Combinatorial optimization problems in wireless switch design ». *4OR*, 5:319–33, 2006a.

R. Sirdey. « A polyhedral approach to reroute sequence planning in MPLS networks ». Rapport technique PE/BSC/INF/20291 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, France, 2006b.

R. Sirdey. « Sudokus et algorithmes de recuit ». *Quadrature*, 62, octobre-décembre 2006c.

R. Sirdey. « Des solutions pour faire bonne figure ». *La Recherche*, avril 2007a.

R. Sirdey. « Multipassive without atomic diffusion ». Rapport technique, service d'architecture BSC, Nortel GSM Access R&D, France, janvier 2007b.

R. Sirdey. « Optimiser... en découpant des polyèdres ». *L'Ouvert*, 115, juillet 2007c.

R. Sirdey. « Sudokus et programmation linéaire ». *Quadrature*, 63, janvier-mars 2007d.

R. Sirdey. « Models and algorithms for the reconfiguration of distributed wireless switching systems ». *4OR*, 6:195–198, 2008.

R. Sirdey, J. Carlier, H. Kerivin et D. Nace. « On a resource-constrained scheduling problem with application to distributed systems reconfiguration ». *European Journal of Operational Research*, 183:546–563, 2007.

R. Sirdey, J. Carlier et D. Nace. « Approximate resolution of a resource-constrained scheduling problem ». *Journal of Heuristics*, 15:1–17, 2009.

R. Sirdey, J. Carlier et D. Nace. « A GRASP for a resource-constrained scheduling problem ». *International Journal of Innovative Computing and Applications*, 2:143–149, 2010.

R. Sirdey et H. Kerivin. « A branch-and-cut algorithm for a resource-constrained scheduling problem ». *RAIRO—Operations Research*, 41:235–251, 2006a.

R. Sirdey et H. Kerivin. « Polyèdres et reconfiguration dynamique d'auto-commutateurs répartis ». Dans *JPOC3*, juin 2006b.

R. Sirdey et H. Kerivin. « Polyhedral combinatorics of a resource-constrained ordering problem part I: on the partial linear ordering polytope ». Rapport technique PE/BSC/INF/017912 V01/EN, service d'architecture BSC, Nortel GSM Access R&D, France, 2006c.

R. Sirdey et H. Kerivin. « Sur le polytope des sous-tournois sans circuit ». Dans *Roadéf*, février 2006d.

R. Sirdey, D. Plainfossé et J.-P. Gauthier. « A practical approach to combinatorial optimization problems encountered in the design of a high availability distributed system ». Dans *Proceedings of the International Network Optimization Conference*, pages 532–539, 2003.

D. J. Songhurst. « Teletraffic engineering ». Dans J. E. Flood, éditeur, *Telecommunication Networks*, tome 36 de *IEE Telecommunications Series*. Institution of Electrical Engineers, London, 1997.

F. Sourd. *Contributions à l'étude et à la résolution de problèmes d'ordonnancement disjonctif*. Thèse de doctorat, Université Pierre et Marie Curie, 2000.

A. Tanenbaum. *Réseaux*. Sciences Sup. Prentice Hall et Dunod, 1996.

F. Tercinet. *Méthodes arborescentes pour la résolution des problèmes d'ordonnancement, conception d'un outils d'aide au développement*. Thèse de doctorat, Université de Tours, 2004.

E. Triki, Y. Colette et P. Siarry. « A theoretical study on the behavior of simulated annealing leading to a new cooling schedule ». *European Journal of Operational Research*, 166:77–92, 2005.

A. Turing. « On computable numbers, with an application to the *Entscheidungsproblem* ». *Proceedings of the Mathematical Society*, 42:230–265, 1936.

A. Turing et J.-Y. Girard. *La machine de Turing*. Sources du savoir. Éditions du Seuil, 1995.

P. J. M. van Laarhoven et E. H. L. Aarts. *Simulated annealing: theory and applications*. Mathematics and its Applications. Kluwer Academic Publisher, 1987.

L. A. Wolsey. « Faces for linear inequality in 0-1 variables ». *Mathematical Programming*, 8:165–178, 1975.

G. Yu. « On the max-min 0-1 knapsack problem with robust optimization applications ». *Operations Research*, 44:407–415, 1996.

Mis en page avec LaTeX, compilation du 27 novembre 2011.