Segmentation et évolution pour la planification : le système Divide-And-Evolve

Jacques Bibai

UNIVERSITÉ DE PARIS-SUD XI - ORSAY

ÉCOLE DOCTORALE D'INFORMATIQUE DE PARIS-SUD

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Paris-sud XI - Orsay
Mention : Informatique

Présentée et soutenue par

Jacques BIBAI

Segmentation et évolution pour la planification : le système Divide-And-Evolve

Thèse dirigée par Marc Schoenauer et Pierre Savéant

préparée à THALES Research and Technology France

Jury :

Rapporteurs :	Jin-Kao Hao	-	LERIA
	Pierre Marquis	-	CRIL-CNRS
Directeurs :	Marc Schoenauer	-	INRIA-Saclay île-de-france
	Pierre Savéant	-	THALES R&T
Président :	Abdel Lisser	-	LRI
Examinateurs :	Olivier Buffet	-	LORIA
	Vincent Vidal	-	ONERA

Remerciements

Ma profonde gratitude s'adresse tout d'abord à mes encadrants Marc Schoenauer et Pierre Savéant, pour leur confiance, les conseils judicieux qu'ils m'ont généreusement prodigués et la passion de la recherche qu'ils m'ont transmise. Leur disponibilité constante et leur rigueur scientifique ont été des atouts majeurs pour la réalisation de cette thèse.

Je remercie Vincent Vidal pour l'intérêt ainsi que toutes les contributions qu'il a porté à ce travail et pour le temps qu'il y a consacré.

Je remercie Colette Pierre et Batard Itamar pour le financement de mes études d'ingénieur.

Ma gratitude va également à Jean-Philippe Famin, Souley Boube, Jean Baptiste Hoock et Mouadh Yagoubi pour la relecture de mon manuscrit de thèse ainsi qu'à Nicolas Museux pour toutes ses remarques pertinentes lors de nos multiples discussions.

Je remercie aussi mes amis, mes anciens collègues de THALES Reserach & Technology France et les membres de l'équipe TAO du Laboratoire de Recherche en Informatique de l'université d'Orsay pour leur accueil, leur sympathie et leurs encouragements. Je tiens à partager le plaisir de présenter cette thèse avec eux.

Je voudrais témoigner ici ma reconnaissance à toute ma famille et amis pour leur soutien. Je pense principalement à mes parents Jean Marcel et Jacqueline, mes frères et soeurs, mes amis de l'Université de Yaoundé I, ceux de l'université de Bretagne Occidentale et ceux de l'ENSIIE. J'exprime également ma gratitude envers mon cousin Christian qui m'a permis de poursuivre mes études en Europe.

Table des matières

Chapitre 1

Introduction

Sommaire

Le raisonnement humain en psychologie cognitive regroupe des activités mentales qui consistent à augmenter l'information disponible en produisant des informations nouvelles à partir d'anciennes. Il existe deux statuts : fiable et correcte qu'on cherche à valider en cherchant d'autres informations qui lui sont liées. Cela permet de comprendre comment on utilise les connaissances à sa disposition ; comment on les transforme ; comment on les organise ; Pourquoi ? : comprendre un message, une situation, chercher la cohérence de phénomènes de cause à effet entre deux phénomènes, pour essayer de prendre des décisions, choisir, résoudre des problèmes selon les contraintes de la situation. Il y a différents types de raisonnements qui se différencient par les mécanismes mis en oeuvre. Le raisonnement conditionnel par exemple nécessite une situation de départ : prémisses ou support du raisonnement. Les prémisses sont correctes du point de vue gramatical et formulées sous forme déclarative. Soit elles attribuent une propriété à un élément, soit précisent une relation entre deux ou plusieurs éléments. Une prémisse fixe le caractère de vérité ou de fausseté d'un élément et peut être associée à des connecteurs (si, alors, et, ou ...) pour former des compositions, et selon la nature du connecteur, la valeur des propositions est différente.

Gestalt [Koffka 1935] dans ses travaux sur la psychologie de la forme, identifie quatres étapes de résolution d'un problème par un homme :

- la préparation : elle consiste à reconnaître le problème existant, les différences entre situation (ou état) initiale et finale qu'il souhaite atteindre, et à comprendre les contraintes posées.
- l'incubation : il cherche à résoudre le problème mais les différentes tentatives sont des échecs, qui vont le mener à laisser le problème de côté.

- l'illumination : il va reprendre le problème après une illumination soudaine, solution *insight*.
- la rectification : elle consiste à vérifier que la solution correspond à la situation finale.

Newell et Simon [Newell 1972] dans les années 70 s'inspirant de la théorie de Gestalt proposent une modélisation de la résolution d'un problème par un homme à travers une simulation informatique. Ce modèle est composé de trois composantes principales : un état initial, un état final et des opérateurs. Un opérateur est un processus de changements de situations ou des mouvements autorisés par des règles traduisant des connaissances expertes du problème. Ces processus vont mener de la situation initiale à la situation finale en passant par des étapes intermédiaires. Ainsi résoudre un problème se résume à construire un espace de problèmes ou une série de situations initiales et situations finales, puis à sélectionner l'opérateur adéquat (ou action) pour chacun des problèmes de la série. La sélection et l'application d'une opération (ou d'une règle) à effectuer va progressivement transformer la situation initiale du problème en un état contenant la situation finale. Cette modélisation suppose que toutes les informations contenues dans l'énoncé suffisent pour résoudre le problème et qu'il est bien défini.

Les informaticiens Fikes et Nilsson [Fikes 1971] formalisent ce modèle à l'aide d'une restriction de la logique du premier ordre. Ils introduisent alors le modèle STRIPS (STanford Research Institute Problem Solver) pour la description des problèmes dit de planification en intelligence artificielle. La planification, discipline de l'Intelligence Artificielle (IA), cherche à concevoir des systèmes capables de générer automatiquement, grâce à une procédure formalisée, un résultat articulé, sous la forme d'un système intégré de décisions appelé plan-solution.

Un état (ou une situation) sera représenté par un ensemble d'atomes et un opérateur (ou une règle ou une connaissance experte) par un triplet d'ensembles d'atomes représentant :

- Les pré-conditions : ce sont les atomes qui déterminent les conditions d'application de l'opérateur. Un opérateur peut être appliqué dans un état si et seulement si toutes ses pré-conditions sont contenues dans l'état.
- Les ajouts : ce sont les atomes qui seront ajoutés à l'état courant après application de l'opérateur.
- Les retraits : ce sont les atomes qui seront retirés à l'état courant après application de l'opérateur.

Afin de permettre une représentation plus concise et plus claire des problèmes de planification, plusieurs améliorations ont été proposées. Ceci a notamment été fait dans le langage ADL (Action Description Language) [Pednault 1989] et récemment dans le langage PDDL (Planning Domain Definition Language) [McDermott 1998, Fox 2003a, Fox 2003b] enrichi dans le cadre des compétitions internationales de planification (IPC-"International Planning Competitions"). Les progrès réalisés ces dernières années permettent au langage PDDL de décrire des domaines complexes et de plus en plus proches de problèmes réels, comportant plusieurs dizaines d'opérateurs et qui prennent en compte certains aspects temporels

et de gestion des ressources.

1.1 Le langage de description PDDL

La définition d'un problème de planification à l'aide de PDDL est composée de deux parties séparées : un domaine générique modélisant la connaissance experte, commun à tous les scénarii ; une description d'un scénario (ou une instance ou un problème).

1.1.1 La définition du domaine

Le domaine contient la déclaration des types (:typing) pour la définition des prédicats, la définition des prédicats (:predicates) et la définition générique des règles expertes ou opérateurs (:action). Le pseudo-code permettant la définition d'un domaine de planification tel que défini par Fikes et Nilsson est :

```
(define (domain DOMAIN_NAME)
  (:requirements [:strips])
  (:typing T1 T2 T3 ... TN - objects)
  (:predicates (PREDICATE_1_NAME [?A1_1 - T1_1 ... ?AN_1 - TN_1])
               (PREDICATE_2_NAME [?A1_2 - T1_2 ... ?AN_2 - TN_2])
       ...)

  (:action ACTION_1_NAME
    :parameters ([?P1 ?P2 ... ?PN])
    :precondition PRECOND_FORMULA
    :effect EFFECT_FORMULA
   )

  (:action ACTION_2_NAME
    ...)

  ...)
```

1.1.2 Définition d'un opérateur

Les paramètres (:parameters) d'un opérateur sont des objets typés ou des instances des prédicats (ou atomes) spécifiés par les préconditions (:precondition) d'application de l'action. Les préconditions contiennent aussi des conjonctions d'atomes. Par exemple, pour une précondition portant uniquement sur un prédicat d'arité 2 :

```
PREDICATE[?A1 -T1 ?A2 - T2]
```

la formule la décrivant sera :

```
(and (T1 ?P1) (T2 ?P2) (PREDICATE (?P1 ?P2))
```

Les effets de l'opérateur sont des conjonctions d'ajouts d'atomes et des conjonctions de retraits d'atomes (cf. exemple ci-dessous). Les retraits d'atomes sont précédés du mot : **not**.

```
(:action ACTION_1_NAME
  :parameters ([?P1 - T1 ?P2 - T2 ... ?PN - TN])
  :precondition PRECOND_FORMULA
  :effect EFFECT_FORMULA
 )
```

1.1.3 Définition d'une instance (ou d'un problème)

Une instance ou un scénario est composé des objets autorisés à être manipulés, une description de l'état initial (ou situation initiale) et une description de l'état final (ou situation finale) qu'on aimerait atteindre. Le pseudo code de la définition d'une instance est :

```
(define (problem PROBLEM_NAME)
 (:domain DOMAIN_NAME)
 (:objects OBJ1 OBJ2 ... OBJ_N)
 (:init ATOM1 ATOM2 ... ATOM_N)
 (:goal CONDITION_FORMULA)
 )
```

Illustrons ces concepts par un exemple simple, le jeu du taquin.

1.1.4 Exemple : le jeu du taquin

Le taquin est un jeu solitaire en forme de damier composé de n carreaux numérotés de 1 à n qui glissent dans un cadre pouvant en contenir $n+1$. Le but du jeu consiste à remettre dans l'ordre les n carreaux à partir d'une configuration initiale quelconque.

La seule connaissance experte (ou règle) ici est le déplacement d'un carreau d'une position donnée à une autre. Les deux objets manipulés dans ce jeu sont les tuiles (tile) et les positions (position). Pour la description d'un état, nous aurons besoin de spécifier la position d'une tuile (at), de décrire le cadre (neighbor) et de définir la position vide (empty). La description du domaine du jeu de taquin par PDDL est :

```
(define (domain n-puzzle-typed)
  (:types position tile)
  (:predicates (at ?tile - tile ?position - position)
               (neighbor ?p1 - position ?p2 - position)
               (empty ?position - position)
   )

  (:action move
```

```
    :parameters (?tile - tile ?from ?to - position)
    :precondition (and (neighbor ?from ?to)
                       (at ?tile ?from)
                       (empty ?to))
    :effect (and (at ?tile ?to) (empty ?from) ;; les ajouts
                 (not (at ?tile ?from)) (not (empty ?to))) ;; les retraits
  )
)
```

Ce domaine sera commun à tous les jeux du taquin et une description d'une instance, par exemple, le jeu du taquin contenant 8 carreaux (cf. figure 1.1), sera :

```
(define (problem n-puzzle-bootstrap-3x3-01)
 (:domain n-puzzle-typed)
 ;;déclaration des objets autorisés
 (:objects p_1_1 p_1_2 p_1_3 p_2_1 p_2_2 p_2_3 p_3_1 p_3_2 p_3_3 - position
  t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 - tile)
 (:init
   ;;positions initiales des tuiles
   (at t_4 p_1_1)
   (empty p_1_2)
   (at t_8 p_1_3)
   (at t_6 p_2_1)
   (at t_3 p_2_2)
   (at t_2 p_2_3)
   (at t_1 p_3_1)
   (at t_5 p_3_2)
   (at t_7 p_3_3)
   ;;definition du cadre
   (neighbor p_1_1 p_1_2)
   (neighbor p_1_2 p_1_1)
   (neighbor p_1_2 p_1_3)
   (neighbor p_1_3 p_1_2)
   (neighbor p_2_1 p_2_2)
   (neighbor p_2_2 p_2_1)
   (neighbor p_2_2 p_2_3)
   (neighbor p_2_3 p_2_2)
   (neighbor p_3_1 p_3_2)
   (neighbor p_3_2 p_3_1)
   (neighbor p_3_2 p_3_3)
   (neighbor p_3_3 p_3_2)
   (neighbor p_1_1 p_2_1)
   (neighbor p_2_1 p_1_1)
   (neighbor p_1_2 p_2_2)
   (neighbor p_2_2 p_1_2)
```

```
    (neighbor p_1_3 p_2_3)
    (neighbor p_2_3 p_1_3)
    (neighbor p_2_1 p_3_1)
    (neighbor p_3_1 p_2_1)
    (neighbor p_2_2 p_3_2)
    (neighbor p_3_2 p_2_2)
    (neighbor p_2_3 p_3_3)
    (neighbor p_3_3 p_2_3))
  (:goal (and
    ;;positions finales des tuiles
    (at t_1 p_1_1)
    (at t_2 p_1_2)
    (at t_3 p_1_3)
    (at t_4 p_2_1)
    (at t_5 p_2_2)
    (at t_6 p_2_3)
    (at t_7 p_3_1)
    (at t_8 p_3_2))))
```

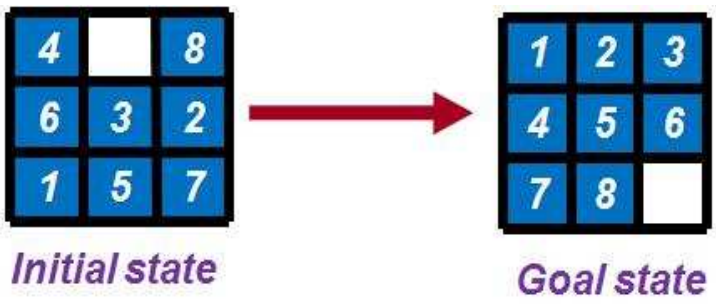

FIG. 1.1 – Jeu de taquin à 8 carreaux

PDDL a été développé dans le but d'une part de standardiser la description des domaines et les problèmes de planification, et d'autre part pour la mise en place de la compétition IPC.

1.1.5 La compétition IPC

La compétition IPC *International Planning Compétition* est une biennale[1] organisée dans le cadre de la conférence ICAPS[2] (*International Conference on Planning and Scheduling*). Les objectifs de la compétition IPC sont de faire progresser et

[1] L'édition 2010 n'a pas eu lieu, faute d'organisateurs.
[2] http ://ipc.icaps-conference.org/

d'analyser l'état de l'art des planificateurs (ou système résolvant les problèmes de planification) génériques, de fournir un forum pour la comparaison empirique des systèmes de planification, de mettre en évidence les défis et limites des capacités actuelles, de proposer de nouvelles orientations pour la recherche, et de créer un formalisme de représentation et un ensemble de benchmarks qui peuvent aider à la comparaison et l'évaluation des systèmes de planification.

La compétition IPC est divisée en trois parties indépendantes :

- la partie dite *Deterministic* qui porte sur les problèmes de planifications observables (ou classique). L'état résultant de l'application d'une action est entièrement connu.
- la partie dite *Uncertainty* qui porte sur les problèmes de planification incertain. Il s'agit ici de gérer des incertitudes (quantifiées par des probabilités ou non) sur les observations et/ou sur les résultats des actions. En pratique les deux principales catégories qui ont été considérées sont la planification conformante et la planification probabiliste.
- la partie dite *Learning* qui évalue les méthodes de résolution des problèmes de planification observables nécessitant une phase d'apprentissage. En d'autres termes, sous l'hypothèse de régularité des instances d'un domaine donné, certaines structures[3] ou méthodes de résolution vont être préalablement apprises par les planificateurs sur quelques instances et utilisées ensuite comme expertise pour résolution de toutes les instances du domaine.

Dans le cadre de cette thèse, nous nous intéresserons uniquement à la résolution générique des problèmes de planification observables. On distingue pour ces problèmes trois grandes familles en fonction du type d'opérateurs utilisés. On parlera de *planification STRIPS classique* lorsque les opérateurs n'auront ni coûts ni durées, de *planification avec coûts* lorsque les opérateurs auront des coûts d'exécution et de *planification temporelle* lorsque les opérateurs auront des durées et leurs instances (ou actions) pourront être concurrentes.

1.1.6 Les benchmarks de la planification observable

Les domaines des benchmarks *Deterministic* de la compétition IPC[4] contiennent plus d'une cinquantaine de domaines qui peuvent être regroupés en quatres grandes catégories : ceux traitant des problèmes de logistique par exemple les domaines *Depots, Driver, Storage,Trucks et ZenoTravel...* ; ceux traitant des missions de robots ce sont par exemple les domaines *satellite, Rovers...* ; ceux traitant des problèmes de jeux comme par exemple *FreeCell, Sokoban, Peg solitaire, n-puzzle, Goldminer...* ; et ceux issus d'applications métiers comme par exemple *Openstacks, Pathways, PipesWorld, crewplanning....*

[3] Par exemple une séquence d'actions applicable à la fois à partir d'un état.

[4] http ://planning.cis.strath.ac.uk/competition/domains.html, http ://zeus.ing.unibs.it/ipc-5/domains.html et http ://ipc.informatik.uni-freiburg.de/Domains

1.2 La recherche d'une solution en IA

En Intelligence Artificielle (IA), la planification suscite actuellement beaucoup d'intérêt, car elle combine les deux domaines majeurs de l'IA : l'exploration et la logique. Un planificateur peut être vu soit comme un programme qui cherche une solution (ou plan), soit comme un programme qui démontre de façon constructive l'existence d'une solution. Le croisement de ces deux domaines a permis à la fois d'améliorer les performances au cours des vingt dernières années et de généraliser l'emploi des planificateurs dans les applications industrielles.

Un critère de qualité, dépendant du type de problème résolu, est associé à chaque solution. Ce sera par exemple, pour le jeu du taquin, le nombre total des action d'un plan. En fonction de ce critère de qualité, on distingue deux types de planificateurs :

- les planificateurs non-optimaux, comme le planificateur qui sera étudié dans le cadre de cette thèse, qui recherche des plans indépendamment de leur qualité
- les planificateurs optimaux, comme le planificateur CPT [Vidal 2004a], qui recherche des solutions optimales.

Les planificateurs du début des années 1970 [Fikes 1971] fonctionnaient avec des séquences d'actions totalement ordonnées ou avec une recherche dans l'espace des états. La décomposition de problème était obtenue en calculant un sous-plan pour chaque atome de l'état final, puis en enchaînant les sous-plans dans un certain ordre. Cette approche, se révèla rapidement incomplète car elle ne permet pas l'entrelacement des actions de différents sous-plans.

Une solution au problème d'entrelacement fut la planification par régression à partir de l'état final : les actions du plan totalement ordonné sont réordonnées de manière à éviter les conflits entre les actions. Elle fut proposée par Waldinger [Waldinger 1975] et utilisée par WARPLAN [Warren 1974].

La détection de conflits [Tate 1975] et la protection des conditions accomplies contre les interférences entre actions [Sussman 1975] figurent parmi les idées sous-jacentes à la notion de planification en ordre partiel. Le planificateur NOAH [Sacerdoti 1975, Sacerdoti 1977], et le système non linéaire de Tate [Tate 1977] sont les pionniers de la construction de plan partiellement ordonnée. La planification en ordre partiel domina la recherche pendant les vingt années qui suivirent. TWEAK [Chapman 1987] était une reconstruction et une simplification logique des travaux de planification de l'époque.

Weld et son équipe développèrent UCPOP [Penberthy 1992], le premier planificateur pour les problèmes exprimés en ADL. Malgré le développement d'heuristiques améliorées pour UCPOP, la planification en ordre partiel perdit de son crédit dans les années 1990, notamment avec l'apparition de méthodes plus rapides telle que GRAPHPLAN [Blum 1995, Blum 1997]. Néanmoins, [Nguyen 2001] suggérèrent qu'elle méritait une réhabilitation et leur planificateur REPOP prit une importance croissante par rapport à GRAPHPLAN et devint compétitif avec les planificateurs en espace d'états les plus rapides.

Avrim Blum et Merrick Furst [Blum 1995, Blum 1997] revitalisèrent le domaine de la planification dans l'espace des états avec le système GRAPHPLAN, qui était

incomparablement plus rapide que les planificateurs en ordre partiel de l'époque. D'autres systèmes de graphe de planification suivirent, dont IPP [Koehler 1997], SGP [Weld 1998] et STAN [Fox 1999]. Le programme UNPOP (1996) de Drew McDermott fut le premier à suggérer une heuristique basée sur un problème relaxé en ignorant les retraits des opérateurs. Le HSP [Bonet 1998], ainsi que ses futurs dérivées [Bonet 1999, Bonet 2001a], innovèrent en mettant en pratique l'exploration dans l'espace d'états. L'outil d'exploration ayant eu le plus grand succès est FF [Hoffmann 2001], vainqueur de la compétition de planification en 2000. Les très performants planificateurs actuels comme YAHSP [Vidal 2004b], Fast Downward [Helmert 2006] (vainqueur de la compétition de planification 2004), LAMA [Richter 2008] (vainqueur de la dernière compétition de planification - 2008) utilisent des heuristiques inspirées par GRAPHPLAN. [Nguyen 2002] donnèrent une analyse approfondie des heuristiques dérivées à partir de graphes de planification. LPG [Gerevini 2003a, Gerevini 2003b], vainqueur de la compétition de planification 2002, explorait des graphes de planification à l'aide d'une technique d'exploration locale. Ces planificateurs seront détaillés dans le chapitre 2.

1.3 Objectifs de la thèse

Les algorithmes actuels de l'état de l'art permettent de résoudre rapidement[5] des problèmes difficiles qui, même s'ils sont issus de benchmarks, sont largement hors de portée de l'humain. Les plans-solutions qu'ils génèrent, bien que non optimaux, comportent jusqu'à plusieurs milliers d'actions.

Les algorithmes évolutionnaires sont des algorithmes d'optimisation stochastiques basés sur une métaphore avec l'évolution biologique. Une population d'individus (ou ensemble de solutions potentielles de l'espace de recherche) est soumise à une succession d'étapes de sélection naturelle (les individus possédant les meilleures valeurs de la fonction objectif sont favorisés) et de variations aveugles (modifications stochastiques sans rapport avec la fonction objectif). Il en résulte au final des individus adaptés, i.e. des solutions quasi-optimales pour la fonction objectif considérée.

Plusieurs approches d'utilisation des algorithmes évolutionaires [Koza 1992, Spector 1994, Muslea 1997, Westerberg 2000, Brié 2005], pour le contrôle de la combinatoire des problème de planification, ont été proposées. Cependant, les résultats obtenus par ces planificateurs étaient très inférieurs au vue de la qualité de ceux de l'état de l'art. Tout récemment, une approche hybride Divide-and-Evolve combinant les techniques de planificaton à celle des algorithmes évolutionnaires a été proposée [Schoenauer 2006, Schoenauer 2007]. Une des limites des planificateurs optimaux tels que CPT [Vidal 2004a] est le passage à l'échelle, tant au niveau mémoire qu'au niveau temps de calcul. Divide-and-Evolve va tenter de surmonter ces limites en générant et optimisant des séries d'états intermédiaires à l'aide d'un algorithme évolutionnaire. Le calcul de la fonction objectif s'obtient en résolvant séquentielle-

[5] quelques dizaines de minutes pour les ordinateurs de bureau actuels

ment les états d'une séquence de l'état initial du problème vers l'état final avec un planificateur embarqué. Une première ébauche de ce schéma original de collaboration entre un algorithme évolutionnaire et le planificateur optimal CPT capable de résoudre des problèmes simples a été établie. Cependant, l'applicabilité et l'efficacité de cette approche n'ont été démontrés que sur trois instances du domaine Zeno[6] (compétition de planification 2000 - voir annexes) avec l'utilisation des paramètres pour la plupart définis manuellement. De plus, les meilleures solutions trouvées par l'algorithme ainsi développé étaient toujours obtenues dans les toutes premières étapes des différentes exécutions.

L'objectif de cette thèse est donc de développer un algorithme générique hybride basé sur ce schéma de collaboration et ainsi explorer en profondeur cette nouvelle approche.

Afin d'atteindre cet objectif, nous allons tout d'abord définir de manière formelle le problème de planification STRIPS en explorant par la suite quelques méthodes de résolution directe, puis les approches de résolution évolutionnaire et enfin l'approche Divide-and-Evolve. Ensuite, nous étudierons plus en détails l'implantation de l'algorithme générique hybride Divide-and-Evolve en explicitant ses différents paramètres et les méthodes de réglage de ceux-ci. Par la suite, nous verrons des cas expérimentaux reprenant les méthodes directes de résolution et l'approche Divide-and-Evolve afin de comparer et évaluer celles-ci. Enfin, nous conclurons nos travaux et présenterons les perspectives.

[6] http ://planning.cis.strath.ac.uk/competition/domains.html

Chapitre 2

Quelques approches de résolution directe

Sommaire

2.1 Définitions

2.1.1 Le problème de planification

Considérons la logique du premier ordre L construite à partir des vocabulaires V_x, V_c et V_p qui dénotent respectivement des ensembles finis disjoints deux à deux

de symboles, de variables, de constantes et de prédicats. Nous n'utiliserons pas de symbole de fonction.

2.1.1.1 Atome

Un atome est une instance d'un prédicat. La valeur de vérité d'un atome a sera notée $\neg a$ s'il est faux et a s'il est vrai.

2.1.1.2 Formule atomique

Une formule atomique est une conjonction d'atomes.

2.1.1.3 Opérateur

Un opérateur o permet de modifier la valeur de vérité d'un atome. C'est un n-uplet $\langle pr, ad, de, dc\rangle$ où pr, ad et de sont des ensembles finis de formules atomiques dénotant respectivement les pré-conditions, les ajouts et les retraits et dc éventuellement la durée (ou le coût) d'application de l'opérateur.

2.1.1.4 Types de problèmes de planification

Les problèmes de planification diffèrent les uns des autres en fonction du type d'opérateur utilisé. On parlera de *planification classique* pour des opérateurs n'ayant ni durée ni coût, de *planification avec coût* lorsque tous les opérateurs ont des coûts d'utilisation et de *planification temporelle* lorsque tous les opérateurs ont des durées et peuvent être appliqués simultanément.

2.1.1.5 Opérateur STRIPS

Un opérateur $o = \langle pr, ad, de, dc\rangle$ sera de type STRIPS si l'ensemble fini pr ne contient pas de négation.

2.1.1.6 Action

Une action o_1 est une instance $o_\theta = \langle pr_\theta, ad_\theta, de_\theta, dc_\theta\rangle$ d'un opérateur o qui est obtenue par l'application d'une substitution θ définie dans le langage L telle que ad_θ et de_θ sont des ensembles disjoints. Nous noterons par $Pre(o_1), Add(o_1)$ et $Del(o_1)$ respectivement les ensembles finis pr_θ, ad_θ et de_θ et $dur\_cout(o_1)$ éventuellement la durée (ou le coût) de l'action o_1 ($dur\_cout(o_1) = dc_\theta$).

Soit A l'ensemble des atomes de valeur de vérité vraie. Notons $\neg A$ l'ensemble des négations des éléments de A et $\Lambda{=}A\bigcup\neg A$ la réunion de A et $\neg A$.

2.1.2 Etat complet, Etat partiel

2.1.2.1 Etat ou ensemble d'atomes cohérent

Un état $E \subseteq \Lambda$ est un sous-ensemble fini d'atomes de Λ tel que la valeur de vérité de la conjonction de tous ses éléments est vraie. En d'autres termes E ne contient pas de paire d'atomes contradictoires (c.-à-d. de valeur de vérité à la fois vraie et fausse).

2.1.2.2 Etat complet

Un état complet $E_c \subseteq \Lambda$ est un sous-ensemble d'atomes cohérents de Λ de cardinalité maximale.

Nous noterons par $card_{max}$ la cardinalité de tout état complet de Λ.

2.1.2.3 Etat partiel

Un état partiel $E_p \subseteq \Lambda$ est un sous-ensemble d'atomes cohérents de Λ et de cardinalité strictement inférieure à $card_{max}$.

Dans la suite un état partiel sera noté s. On notera par $card_s$ le cardinal de l'état partiel s.

2.1.2.4 Exemple : état partiel, état complet

Soit $A = \{a_1, a_2, a_3, a_4\}$ et $\neg A = \{\neg a_1, \neg a_2, \neg a_3, \neg a_4\}$. Alors :

- $s_1 = \{a_1\}$ et $s_2 = \{a_2, a_3, a_4\}$ sont des états partiels.
- $s_3 = \{a_1, \neg a_2, a_3, a_4\}$ et $s_4 = \{a_1, \neg a_2, \neg a_3, a_4\}$ sont des états complets
- $s_5 = \{a_1, \neg a_1\}$ n'est pas un état.

2.1.3 La planification STRIPS

Un problème de planification STRIPS est un n-uplet $\Pi = \langle A, O, I, G \rangle$ où A est l'ensemble des atomes de base, $I \subseteq A$ et $G \subseteq A$ représentent respectivement la situation initiale complète (c.-à-d. un état complet de Λ- les atomes n'apparaissant pas dans I ont pour valeur de vérité : faux) et la situation finale partielle (c.-à-d. un état partiel), et O l'ensemble des actions, instances d'opérateurs de type STRIPS.

2.1.3.1 Solution ou plan

Une solution d'un problème de planification, ou **plan**, est une séquence finie $P = \langle Q_0, Q_1, ..., Q_n \rangle$ d'ensembles finis d'actions. Si $n = 0$, le plan solution P est vide et sera noté $P = \langle angle$.

2.1.3.2 Qualité d'un plan

La qualité d'un plan est fonction du type de problème résolu. Elle sera

- le nombre total d'actions pour les problèmes de planification STRIPS classique (les actions n'ont ni coût ni durée)
- le coût total, somme des coûts des actions du plan pour les problèmes de planification avec coût
- et le makespan (ou la durée totale des actions du plan) pour les problèmes de planification temporelle.

2.1.3.3 Complexité

[Bylander 1994a] montra que la complexité des problèmes de planification STRIPS est PSPACE. [Helmert 2008] étudia plus en détail la complexité des benchmarks des domaines de planification STRIPS classique de la compétition internationales de planification. Il montra que la complexité de ces domaines est PSPACE ou EXPSPACE. [Rintanen 2007] étudia celle des problèmes de planification temporel. Il montra qu'elle est EXPSPACE pour les problèmes de planification temporellement expressifs. Aucun résultat portant sur l'étude de la complexité des problèmes de planification avec coût n'a été publié à notre connaissance.

2.2 GRAPHPLAN

Le planificateur Graphplan [Blum 1995, Blum 1997] transforme un problème de planification en une structure appelée graphe de planification, puis effectue une recherche dans cette structure. Le graphe de planification est un graphe orienté constitué de niveaux successifs contenant deux types de noeuds : les noeuds des atomes et les noeuds d'actions. Le premier niveau contient uniquement les noeuds des atomes appartenant à l'état initial du problème. Le second niveau contient un noeud pour chaque action applicable à l'état initial, et un noeud d'atomes pour chacun de ses effets. Il est composé de trois types d'arcs : ceux liant les pré-conditions d'un niveau aux actions du niveau suivant qui les utilisent, ceux liants les actions et leurs effets positifs (ou ajouts), et ceux liants les actions et leurs effets négatifs (ou retraits). Une action factice (appelée non-opérateur ou no-op) qui a pour précondition un atome et pour effet l'ajout du même atome, est ajoutée afin de résoudre le problème du décor : tout atome appartenant à un état sur lequel on applique un opérateur et qui n'est pas retiré par ce dernier appartient à l'état résultant. L'application d'un no-op signifie que l'atome correspondant n'a pas été retiré par une autre action. Ensuite, les exclusions mutuelles (ou mutex) entre les actions du second niveau, sont calculées. Les mutex entre les actions sont des contraintes binaires représentant l'impossibilité, pour deux actions, d'être appliquées en même temps à ce niveau. Enfin, on calcule les exclusions mutuelles entre les atomes correspondant aux ajouts et aux retraits des actions du second niveau, qui correspondent à l'impossibilité, pour deux atomes, d'être présents dans un même état. Les niveaux suivants sont

calculés en suivant ce schéma, et en tenant compte des exclusions mutuelles entre les atomes du niveau précédent. Le graphe est étendu niveau par niveau, jusqu'à ce que les buts du problème soient présents dans le dernier niveau et ne contiennent pas d'exclusions mutuelles. On tente alors d'extraire une solution par un algorithme de recherche arrière avec retour arrière (ou *Backtracks*). En cas d'échec, le graphe est étendu d'un niveau supplémentaire et la recherche est relancée. Ce processus se poursuit jusqu'à l'obtention soit d'un plan-solution, soit d'une condition exprimant l'impossibilité de trouver une solution au problème.

2.2.1 Définitions : action mutex, atomes mutex, Graphe de planification

2.2.1.1 Indépendance entre actions

Deux actions o_1 et o_2 telles que $o_1 \neq o_2$ sont indépendantes si et seulement si :

$$(Add(o_1) \cup Pre(o_1)) \cap Del(o_2) = \{\} \; et \; (Add(o_2) \cup Pre(o_2)) \cap Del(o_1) = \{\}$$

2.2.1.2 Actions mutex, Atomes mutex

Deux actions o_1 et o_2 d'un même niveau dans le graphe de planification sont mutuellement exclusives (ou mutex) si et seulement si :

- elles ne sont pas indépendantes ou,
- elles ont des pré-conditions mutex au niveau précédent (elles ne peuvent donc pas être déclenchées en même temps) :
  $\exists (p, q) \in Pre(o_1) \times Pre(o_2)$ telles que p et q sont mutex.
  - Deux atomes p et q sont mutex au niveau i si et seulement si tous les couples d'actions qui les produisent à ce niveau sont mutex.
    $\forall o_1, o_2 \mid p \in Add(o_1), q \in Add(o_2)$, o_1 et o_2 mutex

2.2.1.3 No-op d'un atome

Le no-op d'un atome a est l'action dénotée par N_a telle que :

$$Prec(N\_a) = \{a\}, Add(N\_a) = \{a\}, Del(N\_a) = \{\}$$

2.2.1.4 Graphe de planification

Soit $\Pi = \langle A, O, I, G \rangle$ un problème de planification, R une relation binaire sur O et k un entier naturel. *Le graphe de planification d'ordre k* du problème Π et la relation R est le triplet $\langle Noeuds, \; Arcs, \; Mutex \rangle$ défini par : $\langle Noeuds, \; Arcs, \; Mutexangle = ConsGP(\Pi, R, k)$ où $ConsGP$ est l'application définie récursivement par :

- Pour $i = 0$: $ConsGP(\Pi, R, k) = \langle a(0) = \{a \in I\}, \{\}, \{\} \rangle$
- Pour $i \in [1, k]$: soit $\langle Npr, Apr, Mpangle = ConsGP(\Pi, R, i-1)$.
  $ConsGP(\Pi, R, i) = \langle Npr \cup No \cup Na, Apr \cup Ap \cup Aa \cup Ar, Mpr \cup Mo \cup Ma \rangle$, **avec :**
  Noeuds d'actions :

$No = \{o(i)|o \in O, (\forall a \in Prec(o), a(i-1) \in Npr)$, $(\forall \{a,b\} \in Prec(o), \{a(i-1)$, $b(i-1)\} \notin Mpr)\} \cup \{N\_a(i)|N\_a \notin A, a(i-1) \in Npr\}$

Noeuds d'atomes :

$Na = \{a(i)|a \in Add(o), o(i) \in No\}$

Arcs de précondition :

$Ap = \{(a(i-1), o(i))|o(i) \in No, a \in Prec(o)\}$

Arcs d'ajout :

$Aa = \{(o(i), a(i))|o(i) \in No, a \in Add(o)\}$

Arcs de retrait :

$Ar = \{(o(i), a(i))|o(i) \in No, a \in Del(o)\}$

Mutex entre actions :

$Mo = \{\{o_1(i), o_2(i)\}|\{o_1(i), o_2(i)\} \subseteq No, o_1 \neq o_2, (\neg(o_1 R o_2)$

ou $(\exists \{a(i-1), b(i-1)\} \subseteq Mpr, a \in Prec(o_1), b \in Prec(o_2)))\}$

Mutex entre atomes :

$Ma = \{\{a(i), b(i)\}|\{a(i), b(i)\} \subseteq Na, a \neq b, (\forall (o_1(i), a(i)), (o_1(i), b(i)) \in Ao, \{o_1(i), o_2(i)\} \subseteq Mo\}$

2.2.1.5 Exemple

Soit $A = \{a, b, c, d\}$ l'ensemble des atomes ; $O = \{A, B, C\}$ l'ensemble des actions tel que : $Pre(A) = \{a\}$, $Add(A) = \{b\}$, $Del(A) = \{\}$, $Pre(B) = \{a\}$, $Add(B) = \{c\}$, $Del(B) = \{a\}$, $Pre(C) = \{b, c\}$, $Add(C) = \{d\}$ *et* $Del(A) = \{\}$; $I = \{a\}$ et $G = \{d\}$. Le Graphplan construit à partir de ce problème est donné figure 2.1.

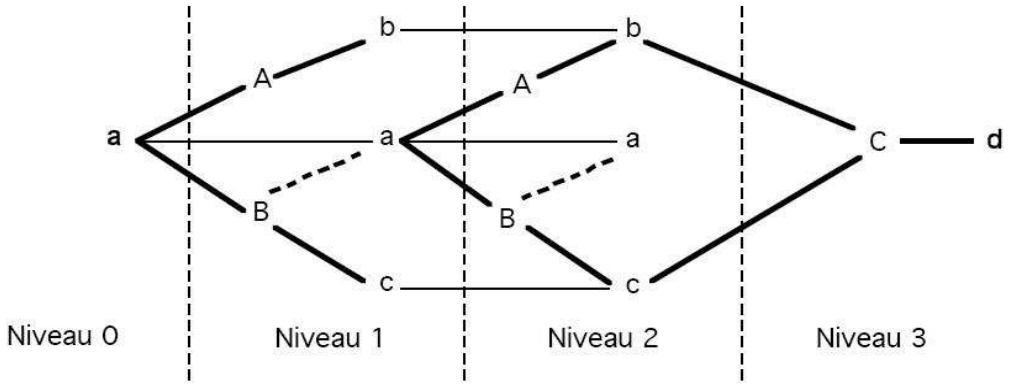

FIG. 2.1 – Exemple Graphplan. Les actions A et B sont mutuellement exclusives car B retire l'atome a qui est une précondition de A. Au niveau 1, les paires de pré-conditions qui sont mutuellement exclusives sont donc $\{a, c\}$ et $\{b, c\}$. On ne peut donc pas utiliser l'action C au niveau 2. Au niveau 2, les atomes b et c ne sont plus mutuellement exclusifs. L'action C peut donc maintenant être appliquée au niveau 3 où elle produit le but.

2.2.1.6 Notations

Soit GP un graphe de planification.
- $Noeuds(GP), NoeudsA(GP)$ dénotent respectivement l'ensemble des noeuds actions et l'ensemble des noeuds atomes de GP.
- $Mutex(GP), MutexA(GP)$ dénotent respectivement l'ensemble des mutex entre actions, l'ensemble des mutex entre atomes de GP.

2.2.2 Complexité

Le principal résultat concernant la complexité de la création d'un graphe de planification est que sa taille et son temps de création sont polynomiaux par rapport à la taille du problème (cf. théorème 1, [Blum 1997], page 288). En effet, peu importe la nature de la relation que l'on choisit pour la création de ce graphe, puisqu'il s'agira toujours d'une relation binaire dont la taille et le temps de calcul sont polynomiaux par rapport à la taille du problème.

2.2.2.1 Théorème : taille du graphe polynomiale [Blum 1997]

Soit $\Pi = \langle A, O, I, G \rangle$ un problème de planification. Soit m le nombre d'actions ($m = card_O$), n le nombre d'atomes qui peuvent apparaître dans un graphe de planification ($n = card_{Add(O) \cup Del(O) \cup I}$), et p le nombre d'atomes de l'état initial ($p = card_I$). Pour toute relation binaire R sur les actions dont la taille et le temps de calcul sont polynomiaux par rapport à la taille du problème, la taille d'un graphe de planification d'ordre k et le temps de création de ce graphe sont polynomiaux en n, p et m.

schéma de preuve La preuve donnée dans [Blum 1997] ne fait pas intervenir la nature de la relation R entre les actions : il suffit de remarquer que sa taille et son temps de calcul sont polynomiaux par rapport à la taille du problème.

2.2.3 Stabilisation

Lors de la construction du graphe de planification, on pourrait être confronté au phénomène de stabilisation du graphe de planification. En effet, dans un cadre de planification STRIPS classique, on observe le fait suivant : à partir d'un niveau k, tous les niveaux suivants du graphe de planification sont identiques (tous les niveaux j tels que $janglek$ sont égaux). Par égaux, nous entendons qu'ils possèdent le même ensemble d'atomes, le même ensemble d'actions, et les mêmes mutex sur les atomes et sur les actions. On dira alors que la construction du graphe de planification s'est stabilisée au niveau k.

Définition : niveau de stabilisation du graphe de planification Soit GP un graphe de planification d'ordre k. Le niveau de stabilisation de GP est le plus petit

niveau $i\langle k$ tel que :

$$\{a|a(i) \in NoeudsA(GP)\} = \{a|a(i+1) \in NoeudsA(GP)\}$$

$$\{\{a,b\} \in \{a(i),b(i)\} \in MutexA(GP)\} = \{\{a,b\} \in \{a(i{+}1),b(i{+}1)\} \in MutexA(GP)\}$$

Lorsque GP a atteint son niveau de stabilisation, on dira qu'il est stabilisé.

2.2.3.1 Théorème : stabilisation du graphe, [Blum 1997]

Soient $\Pi = \langle A, O, I, G\rangle$ un problème de planification et R une relation binaire sur les actions. Il existe un niveau k tel que, $\forall ianglek, GP = ConsGP(\Pi, R, i)$ est stabilisé et :

$$\{n|n(i) \in Noeuds(GP)\} = \{n|n(i+1) \in Noeuds(GP)\}$$

$$\{\{m,n\}|\{m(i),n(i)\} \in Mutex(GP)\} = \{\{m,n\}|\{m(i{+}1),n(i{+}1)\} \in Mutex(GP)\}$$

schéma de preuve La preuve donnée dans [Blum 1997] ne fait pas intervenir la nature de la relation R entre les actions. Cette preuve est basée sur le fait que si un noeud d'un atome ou d'une action est présent dans le graphe à un niveau donné, alors il sera présent dans tous les niveaux suivants ; et si une exclusion mutuelle entre deux noeuds d'atomes ou d'actions disparaît à un niveau donné, alors elle ne sera plus présente dans les niveaux suivants. Comme la relation R employée porte sur les actions et non sur les noeuds d'actions, cette propriété reste vraie, donc le graphe atteint bien un niveau de stabilisation.

2.2.4 Inertie des atomes

Graphplan travaille donc sur une structure entièrement propositionnelle : les atomes et les actions sont totalement instanciés par les constantes du domaine. L'instanciation des opérateurs peut être faite à chaque extension du graphe, en recherchant tous les unificateurs possibles des pré-conditions des opérateurs avec les atomes obtenus au niveau précédent. Mais comme une action qui apparaît à un niveau reste présente dans tous les niveaux suivants, la recherche des unificateurs qui ont donné ces actions est redondante à chaque extension du graphe. En effet, les prédicats, et par extension les actions, concernent en général une partie seulement des objets ; beaucoup d'instanciations possibles ne sont donc pas pertinentes. Une méthode simple pour instancier efficacement des domaines non typés consiste à repérer les atomes présents dans l'état initial d'un problème et qui sont absents des effets des opérateurs [Weld 1999]. On peut considérer que ces atomes sont inertes. Les atomes formés avec ces prédicats seront présents dans tous les états et contraindront l'instanciation des opérateurs. On peut donc ne calculer que les instanciations des opérateurs incompatibles avec ces atomes. De plus, ces derniers ne jouent aucun rôle dans la construction du graphe de planification, puisqu'ils sont présents dans tous les niveaux et ne prennent jamais part aux exclusions mutuelles ; on peut

donc les supprimer de l'état initial et des pré-conditions des actions qu'ils ont permis d'instancier. On peut noter que ces prédicats ne se limitent pas aux prédicats unaires dont on se sert souvent pour typer implicitement les constantes d'un domaine. Une généralisation de cette méthode est proposée dans [Koehler 1999] pour le langage ADL [Pednault 1989], en considérant aussi l'inertie des atomes qui apparaissent après l'instanciation. En effet, quand on instancie les actions partiellement instanciées grâce à l'inertie des prédicats, on peut encore trouver des actions non pertinentes ou des actions ayant des effets conditionnels non pertinents. Ces actions non pertinentes (ou bien les effets conditionnels non pertinents) peuvent aussi être supprimées, grâce à une étude sur l'inertie des atomes qui les constituent, en suivant le même principe que pour l'inertie des prédicats. Une méthode plus fine d'analyse des types d'atomes est proposée dans l'outil d'analyse TIM ("Type Inference Module") [Fox 1998]. Les constantes et les variables des opérateurs sont typées par l'analyse d'automates finis que l'on peut construire à partir de la description du domaine : les symboles de prédicats sont les états ; les opérateurs constituent les transitions entre les états ; les constantes sont les informations véhiculées par les transitions. Grâce à cette analyse, on peut inférer des types plus complexes que ceux trouvés par l'inertie des prédicats (ou des atomes).

2.2.5 Le calcul des mutex

[Smith 1999] et [Fox 1999] proposent une construction optimisée du graphe de planification, sous la forme d'une structure circulaire à deux niveaux. Cette construction est basée sur la propriété de monotonicité des actions, des atomes et des exclusions mutuelles : un noeud d'atomes ou d'actions apparaissant à un niveau i du graphe de planification reste présent dans tous les niveaux supérieurs du graphe, et une mutex disparaissant à un niveau i du graphe n'apparaîtra plus dans les niveaux supérieurs. Cette propriété permet de construire le graphe de planification sous la forme d'une structure formée d'un seul ensemble de noeuds d'actions et d'un seul ensemble de noeuds d'atomes ; pour chacun de ces noeuds, on conserve le niveau d'apparition de chaque action ou de chaque atome. Pour chaque mutex, on conserve le niveau le plus élevé dans lequel on la rencontre. Les noeuds d'actions sont reliés aux noeuds d'atomes qui correspondent à leurs pré-conditions, leurs ajouts et leurs retraits. Pour chaque noeud d'atomes, on conserve la liste des actions qui l'utilisent, le produisent ou le retirent à chaque niveau. Cette construction est optimisée dans le planificateur STAN ("STate Analysis") [Fox 1999] par l'utilisation de tableaux de bits.

Une telle construction du graphe de planification permet de simplifier le calcul des mutex (cf. [Smith 1999]) à l'aide de leur catégorisation : certaines mutex seront toujours présentes (par exemple pour des actions qui ont des effets contradictoires) alors que d'autres ne le seront pas. En effet, avec la disparition d'une mutex entre deux atomes, disparaît une des raisons qui fait que deux actions sont mutuellement exclusives. La mutex entre ces deux actions peut donc disparaître, si c'était sa seule raison d'être. Ainsi, certaines mutex ne seront jamais recalculées (celles qui sont

permanentes), c'est-à-dire celles que l'on trouve à tous les niveaux du graphe, alors que d'autres le seront. Une fois que ces dernières auront disparu, il sera inutile de les recalculer.

Un autre moyen pour analyser les mutex vient du calcul de types effectué par la méthode de [Fox 1998]. Cette analyse de types permet non seulement d'accélérer l'instanciation des actions, mais aussi de déduire des invariants d'état qui peuvent être employés de plusieurs manières. Par exemple dans le domaine du Monde des cubes classique (http ://users.cecs.anu.edu.au/ jks/bw.html), TIM peut déduire que l'on ne peut pas avoir deux cubes posés sur seul cube ; ce qui sera détecté par Graphplan comme étant une mutex. Ainsi, les invariants d'états trouvés par TIM contiennent entre autres toutes les mutex que trouve Graphplan qui sont permanentes. Graphplan doit recalculer pour chaque niveau ces mutex, quand il ne s'agit pas d'une relation syntaxique entre des actions (intersection des ajouts d'une action et des retraits d'une autre). De plus, il est possible de détecter des mutex permanentes que Graphplan ne peut pas détecter [Fox 2000]. Par exemple, si trois actions mutuellement exclusives deux à deux de façon permanente produisent deux atomes (deux de ces actions en produisant un, la troisième produisant les deux), Graphplan ne peut pas détecter que les deux atomes sont mutuellement exclusifs. [Fox 2000] exhibent un domaine où cela se produit et où un invariant d'état est déduit, qui permet l'ajout d'une mutex que Graphplan ne trouve pas.

En résumé, le calcul des mutex n'est pas une tâche triviale. Néanmoins, plusieurs heuristiques ont été proposées afin d'approcher ce calcul, que nous détaillerons dans la section suivante.

2.3 Les heuristiques

La planification par recherche heuristique a prouvé qu'elle était un cadre de recherche intéressant pour la planification non-optimale de type STRIPS, surtout depuis l'arrivée de planificateurs capables de dépasser, dans la plupart des domaines, les performances des planificateurs tels que Graphplan [Blum 1995, Blum 1997], Blackbox [Kautz 1999], STAN [Fox 1999], SGP [**?**], SGPLAN [Chen 2006, Wah 2006]... C'est l'une des raisons pour laquelle l'attention de la communauté scientifique de planification s'est tournée vers ce cadre de recherche, plus prometteur en terme de performances pour la planification non-optimale. Les heuristiques permettent aussi d'estimer les dates au plus tôt d'apparition d'un atome dans toute solution d'un problème de planification (cf. chapitre 5 section 5.1.1.3).

2.3.1 Pertinence

La comparaison entre deux algorithmes de recherche heuristiques se fonde sur la mesure de **pertinence**. La **pertinence** est définie par le rapport entre le nombre de noeuds développés et la qualité de la solution obtenue.

2.3.2 Heuristique admissible, non admissible, bien informée

Une heuristique sera dite **admissible** (respectivement **non admissible**) si le coût optimal du plan obtenu après application d'une action (ou l'apparition d'un atome) est toujours supérieure (respectivement toujours inférieure) à l'estimation donnée par l'heuristique.

Une heuristique sera dite **bien informée** si l'estimation du coût d'application d'une action (ou d'apparition d'un atome) est proche du coût optimal.

La planification peut être naturellement formulée comme un problème de recherche dans les espaces d'états : à partir d'un état initial, d'un ensemble d'actions qui représentent les transitions entre les états, et d'un but qui caractérise la partie commune de l'ensemble des états que l'on veut atteindre, on recherche une séquence d'actions dont l'application à partir de l'état initial permet d'atteindre un des états buts. Pour cela, on dispose de tous les algorithmes classiques de recherche dans les espaces d'états : largeur d'abord, profondeur d'abord, A*, WA*, IDA*...

Cependant, les algorithmes de résolution générique optimaux, comme par exemple A* [Hart 1968], étant de complexité exponentielle, il est généralement plus judicieux de faire appel à des méthodes heuristiques pour la résolution des problèmes difficiles. L'idée principale pour l'élaboration d'une heuristique est de simplifier (ou relaxer) les données du problème initial afin d'en estimer les coûts des actions ou des atomes. Le problème de planification relaxé est obtenu du problème de planification initial par suppression des effets négatifs (ou retraits) des actions. L'utilisation du problème relaxé pour le calcul d'une heuristique rend l'estimation des coûts admissible et bien informée. Cependant, cette estimation est très complexe à calculer. Néanmoins, à cause de la nature spécifique des problèmes de planification, qui sont souvent composés de sous-problèmes que l'on peut résoudre indépendamment les uns des autres, on peut supposer l'indépendance[1] des atomes de l'état final et admettre qu'ils sont réalisables l'un après l'autre par des plans qui n'interfèrent pas entre eux.

2.3.3 Les heuristiques déduites du problème relaxé : $h^+, h^{add}, h^{max}, h^{FF} \ldots$

L'utilisation des heuristiques déduites du problème de planification relaxé a largement dominé les compétitions internationales de planification (IPC). Par exemple, elle a été la clef des systèmes FF [2] [Hoffmann 2001], Fast Downward[3] [Helmert 2006], et LAMA[4] [Richter 2008]. Cependant, calculer le plan optimal h^+ du problème de planification initial relaxé est encore un problème NP [Bylander 1994b].

[1] Cette hypothèse est rarement justifiée. Contre exemple : l'anomalie de Sussman
[2] Vainqueur de l'édition 2
[3] Vainqueur de l'édition 4
[4] Vainqueur de l'édition 6

[Bonet 1997] proposent une méthode polynomiale d'estimation de l'heuristique, pour le problème relaxé. Cette heuristique estime la longueur d'un plan minimal qui produit un ensemble d'atomes, en définissant le coût d'un atome a à partir d'un état s par une fonction h_s. Cette fonction ajoute 1 au minimum des coûts des ensembles de pré-conditions des actions qui produisent a. Le coût d'un ensemble d'atomes est alors défini à l'aide d'une fonction d'agrégation des coûts des atomes qui le composent. Le coût d'un atome peut être défini récursivement, pour un état s et un atome a, par :

$$h_s(a) = \begin{cases} 0 \text{ si } a \in s \\ i+1 \text{ si } min_{o \in O; a \in Add(o)}[H_s(Pre(o))] = i \\ \infty \text{ sinon} \end{cases} \quad (2.1)$$

où H_s est l'heuristique calculant le coût d'un ensemble d'atomes A_{Pre}.

Plusieurs possibilités peuvent ainsi être envisagées pour H_s : l'heuristique des systèmes ASP [Bonet 1997], HSP [Bonet 1998], HSPr [Bonet 1999] et HSP2 [Bonet 2001a] additionnent le coût de chaque atome de l'ensemble A_{Pre} :

$$H_s^{add}(A_{Pre}) = \sum_{a \in A_{Pre}} h_s(a) \quad (2.2)$$

[Hoffmann 2001] proposent une amélioration de cette heuristique additive basée sur le fait que la résolution du problème relaxé par Graphplan est polynomiale. En effet, puisque les retraits des actions ont été suprimés, le graphe de planification ne contient aucune exclusion mutuelle, donc la procédure d'extraction de la solution est triviale (sans aucun retour arrière). Le coût d'un ensemble d'atomes est alors constitué par la somme des actions du plan parallèle qui produit ces atomes :

$$H^{FF}(A_{Pre}) = \sum_{i \in [0,k]} |O_i| \quad (2.3)$$

où $O_i \in O$ est une action de niveau i du Graphplan.

Elle prend donc en compte les effets positifs des actions, contrairement à l'heuristique additive qui suppose que tous les sous-buts sont indépendants. L'extrême performance du planificateur FF ("Fast Forward") lors de la 3^{ieme} compétition internationale de planification a inspiré la définition de plusieurs de ces variantes telles que $h^{FF/a}$ et h^{sa} [Keyder 2008].

Cependant, ces heuristiques ne sont pas admissibles, car elles surestiment le nombre d'actions nécessaires pour résoudre le problème relaxé. Une possibilité pour les rendre admissible est de remplacer $\sum$ des équations précédentes par le maximum du coût de chaque atome de l'ensemble A_{Pre} [Bonet 2001b], mais celle-ci n'est pas très informative :

$$H_s^{max}(A_{Pre}) = max_{a \in A_{Pre}} h_s(a) \quad (2.4)$$

2.3.4 La famille d'heuristiques h^m

La famille d'heuristiques h^m [Haslum 2000] issues de h^{max} estiment le coût de réalisation d'un ensemble d'atomes A' par le coût du plus coûtant sous-ensemble $A_s \subseteq A'$ tel que $card_{A_s} = m$. Cette famille d'heuristiques est admissible et polynomiale. La précision de l'heuristique h^m est fonction de la valeur de m. Ainsi, pour m grand, h^m est plus précis mais également plus coûteuse. L'heuristique h^m est définie par relaxation à partir de l'heuristique exacte h^* définie par :

$$h^*(A') = \begin{cases} 0 \text{ si } A' \subseteq I \\ \\ min_{E=\{a \in A \mid o \in Plan(\langle A,O,I,A' angle) \wedge a \in Add(o)\}}[cout(o) + h^*(E)] \text{ sinon} \end{cases} \tag{2.5}$$

Avec $Plan(\langle A, O, I, A' angle)$ la solution du problème de planification $\Pi = \langle A, O, I, A' \rangle$.

L'heuristique h^m, qui produit une estimation admissible de h^*, est alors définie par :

$$h^m(A') = \begin{cases} 0 \text{ si } A' \subseteq I \\ \\ min_{E=\{a \in A \mid o \in Plan(\langle A,O,I,A' angle) \wedge a \in Add(o)\}}[cout(o) + h^*(E)] \text{ si } card_{A'} \leq m \\ \\ max_{A_s \subseteq A;\, card_{A_s}=m}[h^m(A_s)] \text{ sinon} \end{cases} \tag{2.6}$$

L'heuristique h^m approxime donc le coût d'un ensemble d'atomes A' tel que $card_{A' anglem}$ par le coût du plus couteux sous ensemble d'atomes $A_s \subseteq A'$ de taille m.

2.3.4.1 Cas spécial $m = 1$: $h^1 = h^{max}$

Pour $m = 1$ par exemple, l'équation ci-dessus devient :

$$h^1(A') = \begin{cases} 0 \text{ si } A' \subseteq I \\ \\ min_{E=\{a \in A \mid o \in Plan(\langle A,O,I,A' angle) \wedge a \in Add(o)\}}[cout(o) + h^*(E)] \text{ si } card_{A'} \leq 1 \\ \\ max_{A_s \subseteq A';\, card_{A_s}=1}[h^1(A_s)] \text{ sinon} \end{cases} \tag{2.7}$$

En d'autres termes :

$$h^1(A') = \begin{cases} 0 \text{ si } A' \subseteq I \\ \\ min_{o \in O;\, a \in Add(o)}[cout(o) + h^1(Pre(o))] \text{ si } A' = \{a\} \\ \\ max_{a \in A'}[h^1(\{a\})] \text{ sinon} \end{cases} \tag{2.8}$$

Ce qui est exactement l'équation de l'heuristique h^{max} (cf. section 2.3.3).

2.3.4.2 Cas spécial $m = 2$: heuristique Max-Paire (algorithme Graphplan)

Pour $m = 2$ l'équation définissant h^m devient :

$$h^2(A') = \begin{cases} 0 \text{ si } A' \subseteq I \\ \\ min_{o \in O, a \in Add(o)}[cout(o) + h^2(Pre(o))] \text{ si } A' = \{a\} \\ \\ min\{min_{o \in O;\ a,b \in Add(o)}[cout(o) + h^2(Pre(o))], \\ \quad min_{o \in O;\ a \in Add(o)}[cout(o) + h^2(Pre(o) \cup \{b\})], \quad \text{si } A' = \{a, b\} \\ \quad min_{o \in O;\ b \in Add(o)}[cout(o) + h^2(Pre(o) \cup \{a\})], \\ \quad min_{o_1, o_2 \in O;\ a \in Add(o_1) \wedge b \in Add(o_2)}[1 + h^2(Pre(o_1) \cup Pre(o_2))] \text{ si parallelisme}\} \\ \\ max_{\{a,b\} \subseteq A'}[h^2(\{a, b\})] \text{ si } card_{A'} angle 2 \end{cases} \tag{2.9}$$

Cette heuristique h^2 est équivalente à l'heuristique du système Graphplan [Blum 1997] qui donne l'index du premier niveau du Graphplan contenant tous les atomes de A' non mutex. On en déduit que si deux actions ont des coûts égaux (ou la même valeur calculée par h^2) et ne sont pas indépendantes alors elles sont mutex et ne peuvent donc pas être exécutées en même temps [Haslum 2000].

2.3.5 Les heuristiques dites *Landmarks* : h^L

Les heuristiques dites *landmarks* [Koehler 2000, Hoffmann 2004], ont pour but d'estimer un ordre entre les atomes nécessaires (ou *landmarks*) dans toutes les solutions possibles d'un problème de planification. Un atome est dit nécessaire s'il est ajouté par au moins une action quelle que soit la solution du problème. La recherche de ces types d'atomes n'est pas une tâche triviale : complexité PSPACE-difficile [Porteous 2001]. L'idée est donc d'approximer cette recherche en utilisant le problème initial relaxé à l'aide du *Landmarks Generation Graph* qui est un dérivé du Graphplan (cf. [Porteous 2001] pour plus de détails).

Pour un ensemble donné d'atomes A', une heuristique dite *landmarks* possible h^L peut être définie par[5] :

$$h^L = n - m + k$$

où n est le nombre de total de *landmarks* estimé, m le nombre de *landmarks* estimé contenu dans A' et k le nombre de *landmarks* n'appartenant pas à A' et étant des pré-conditions directes d'une action ajoutant un *landmarks*.

Plusieurs autres variantes peuvent être trouvées dans [Karpas 2009]. Ces heuristiques sont généralement combinées avec les heuristiques précédentes afin de préférer certaines actions lors de la recherche d'une solution et ainsi augmenter l'efficacité de l'algorithme de recherche.

[5] Cette heuristique est utilisée dans le systèmes LAMA état de l'art des planificateurs approchés non temporel

2.4 Quelques planificateurs

2.4.1 Le planificateur FF

Le planificateur FF[6] ou *Fast Forward* [Hoffmann 2001] est un planificateur heuristique non-optimal de recherche dans l'espace des états (un noeud de son arbre de recherche est un état et une action est un arc). Le nombre d'actions du plan relaxé pour un état constitue pour FF la valeur numérique de l'heuristique pour cet etat, et représente une estimation de la longueur du plan nécessaire pour atteindre les buts (ou l'état final). Une autre information dérivée du graphe de planification relaxé et de sa solution par FF est : l'ensemble des actions dites "utiles". Les actions "utiles" sont les actions du graphe relaxé éxécutables dans l'état pour lequel on souhaite calculer sa valeur heuristique, augmentées dans FF par toutes les actions applicables (ou ayant au moins une précondition) dans cet état et produisant des atomes qui ont été déterminés comme étant des sous-buts au premier niveau du graphe de planification lors de l'éxécution du plan. Ces actions permettent à FF de concentrer ses efforts sur les branches les plus prometteuses, en oubliant les actions qui ne sont pas "utiles" par une variation de son algorithme de recherche locale ("hill-climbing"). Lorsque ce dernier ne parvient pas à trouver une solution, la recherche est relancée depuis le début par un algorithme de recherche complet de type meilleur d'abord. Le bénéfice pouvant être apporté par les actions utiles et la recherche locale est alors perdu.

2.4.2 Le planificateur YAHSP

Les résultats sur de nombreux domaines des benchmarks de la compétition IPC montrent que l'heuristique FF permet d'obtenir une bonne qualité des plans relaxés (cf. http ://ipc.icaps-conference.org/). [Vidal 2004b] constate que le début de ces plans peut souvent être étendu en des plans solutions du problème initial (non relaxé), et qu'un grand nombre d'actions présentes dans ces plans relaxés se trouvent également dans les plans solutions. Il propose donc d'ajouter à la liste des noeuds pouvant être développés à partir d'un état donné des nouveaux noeuds (ou *noeuds anticipés*) issus de l'application d'un certain nombre d'actions du plan relaxé qui produisent le début d'un plan valide. L'état obtenu après application de ces actions est appelé *état anticipé*. Pour le calcul des états anticipés, il suggère d'utiliser autant que possible les actions du plan relaxé et de calculer ces états pour chaque nouveau noeud choisi. Le choix du noeud à développer sera toujours guidé par une heuristique avec une préférence comme dans FF pour les noeuds "utiles". L'heuristique utilisée dans le planificateur YAHSP[7] est une variante de l'heuristique h^{FF}. Les différences viennent d'une part du fait que YAHSP considère qu'une action d'un niveau du Graphplan relaxé peut utiliser, contrairement à FF, comme précondition les ajouts d'une action du même niveau du Graphplan. Et d'autre part, lors de la construction d'une solution d'un problème de planification par FF, les actions sont

[6] FF a été récompensé par le premier prix à la compétition IPC-2
[7] YAHSP a été récompensé par le second prix à la compétition IPC-4

placées dans l'ordre dans lequel elles ont été sélectionnées tandis que dans YAHSP, une action sélectionnée est insérée dans une liste ordonnée d'actions déjà sélectionnée. L'algorithme de recherche est un algorithme classique de type meilleur-d'abord. Bien que les états anticipés ne sont généralement pas les états buts, YAHSP permet généralement de résoudre très rapidement beaucoup plus de problèmes que FF avec des qualités de solutions comparables à ceux de FF [Vidal 2004b].

2.4.3 Le planificateur LAMA

Le planificateur LAMA [Richter 2008] est le vainqueur de la dernière compétition internationale de planification[8]. Il est l'état de l'art actuel des planificateurs approchés non-temporels.

LAMA est un planificateur heuristique incrémental non-optimal de recherche dans l'espace des états. Il est inspiré des planificateurs *FF* [Hoffmann 2001] et *Fast Downward*[9] [Helmert 2006]. LAMA combine une variante de l'heuristique FF ($FF(h_a)$[10]) [Keyder 2008] à l'heuristique dite *Landmarks* h^L. L'heuristique $FF(h_a)$ remplace la somme des actions d'un niveau donné $\sum_{i\in[0,k]} |O_i|$ de l'équation 2.3 par la somme des coûts des actions du même niveau $cost = \sum_{i\in[0,k]} cost(O_i)$. Comme dans FF les actions "utiles" sont préférées aux autres actions, ensuite parmi ces actions "utiles", LAMA choisit en priorité les actions ayant la meilleure heuristique h^L. Son algorithme de recherche est de type WA^* avec $Wangle1$[Pohl 1970]. Le facteur W permet de contrôler la qualité de la solution obtenue. Ainsi plus W sera grand, plus la qualité des solutions obtenues sera mauvaise. Cette spécificité permettant de choisir, de rechercher des solutions de plus ou moins bonne qualité et rend WA^* intéressante pour une recherche incrémentale de solutions [Hansen 2007].

2.4.4 Le planificateur TFD

Le planificateur *Temporal Fast Downward* ou TFD est l'état de l'art des planificateurs temporels non-optimaux [Eyerich 2009]. C'est un planificateur temporel heuristique travaillant dans l'espace des états. Son espace de recherche est représenté par un Domain Transition Graph (DTG). Le DTG est un graphe orienté dont les noeuds sont des états et les arcs, les transitions entres états. Un arc entre deux états A_1 et A_2 du graphe représente une action ayant une précondition dans A_1 et des effets dans A_2. Une valeur réelle correspondant à la date au plus tôt de création d'un état est associée à chaque noeud du DTG correspondant. La recherche d'une solution se fait par une exploration systématique de l'espace à l'aide d'une stratégie gloutonne de type meilleur-d'abord, guidée par une heuristique additive h^{add} adaptée aux actions avec durées[11]. La valeur heuristique d'un état (ou d'un noeud) du

[8] cf. http ://ipc.icaps-conference.org/

[9] vainqueur de la quatrième compétition IPC

[10] L'heuristique h^{FF} adapté au coût.

[11] La somme des coûts d'un ensemble d'atomes de l'équation 2.2 est remplacée par la date au plus tôt de création de l'ensemble d'atomes

DTG est obtenue en appliquant l'algorithme de Dijkstra [Dijkstra 1971] sur le DTG du problème initial relaxé.

2.4.5 Le planificateur LPG

Le planificateur LPG [Gerevini 2003a, Gerevini 2003b] est un planificateur stochastique de recherche dans l'espace des plans partiel. Il est l'un des meilleurs planificateurs temporels actuels et a été le vainqueur de la 3^{ieme} édition de la compétition IPC. Il transforme un problème de planification en une structure appelée Linear Action graph (ou *LA-graph*) qui dérive du Temporal Action graph (ou *TA-graph*), puis effectue une recherche locale dans cette structure. Le *TA-graph* est un graphe orienté de type Graphplan contenant des noeuds d'actions et des noeuds d'atomes. Comme pour Graphplan, chaque niveau du *TA-graph* est composé de noeuds d'atomes et de noeuds d'actions. La connexion entre ces noeuds est fonction soit des pré-conditions des actions pour les arcs entre les noeuds d'atomes et les noeuds d'actions ; soit des effets positifs des actions pour les arcs liants les noeuds d'actions et les noeuds d'atomes ; soit de la relation mutex entre les actions uniquement. Cependant, la prise en compte des atomes mutex est reportée sur les actions avec l'ajout de l'opérateur no-op. Chaque niveau du *TA-graph* peut donc contenir un ensemble d'atomes, d'actions et de no-op provenant des atomes non utilisés par les actions du même niveau. Pour la résolution du problème, ce graphe sera complété par un ensemble de contraintes d'ordre et d'exclusion mutuelle et par un ensemble de valeurs assignées aux actions et aux atomes calculés par une heuristique.

Le *TA-graph* peut être étalé linéairement en autorisant au plus une action par niveau. Ce nouveau graphe, qui porte le nom de Linear Action graph, est le coeur de l'algorithme LPG. La recherche locale consiste alors à insérer ou à retirer une action -le meilleur voisin- dans le *LA-graph* dans le but de résoudre une contrainte d'exclusion mutuelle entre une action et un ou plusieurs no-op. Le meilleur voisin est déterminé par une heuristique qui est une combinaison linéaire du coût d'exécution de l'opération -insertion ou retrait d'une action-, de la durée maximale du plan résultant après application du meilleur voisin et du coût de l'opération.

2.4.6 Le planificateur CPT

Le planificateur temporel CPT[12] [Vidal 2004a] ou "Constraint Programming Temporal" est un planificateur optimal travaillant dans l'espace des plans partiels à l'aide d'une formulation en Partial Order Causal Link (POCL) [McAllester 1991], qui fournissent des règles de branchement intéressantes et d'élagages puissantes et saines basées sur une formulation en contraintes. L'originalité de CPT est l'intègration des heuristiques existantes (la famille h^m) avec les règles de propagation de la formulation en POCL.

[12] Le planificateur CPT a été récompensé par le deuxième prix à la compétition IPC en 2004 (4^{ime} édition) - http ://zeus.ing.unibs.it/ipc-5

Les variables sont définies a priori, pour chaque action, et un ensemble de variables est créé pour chaque précondition de chaque action. Les contraintes correspondent essentiellement à des disjonctions, des implications, des contraintes temporelles, et à leurs combinaisons. Elles sont exprimées pour toutes les actions et pour toutes les pré-conditions. Il en résulte ainsi des contraintes de liens causaux et des contraintes d'ordre liant des actions entre elles. Deux actions admettent une contrainte de causalité si l'exécution de l'une entraîne une possibilité d'exécution de l'autre. Les contraintes d'ordre permettent leur ordonnancement temporel. Toutes les décisions de branchement génèrent des divisions binaires et les nœuds dans l'arbre de recherche correspondent à des plans partiels. Les deux critères de sélection des actions sont les dates de début au plus tôt et la fenêtre temporelle pendant laquelle les actions peuvent s'exécuter. La recherche d'une solution optimale se fait par dichotomie à partir d'une borne minimale calculée à l'aide d'une heuristique de la famille h^m. La stratégie de parcours utilisée ici est celle d'une recherche arborescente en profondeur d'abord de type *generate and test* (algorithme avec **backtrack**). La preuve d'optimalité est obtenue par le parcours complet de l'arbre de recherche à partir du sommet, connaissant une solution en élaguant les branches de l'arbre conduisant à des solutions de qualité inférieure.

2.5 Conclusion

La plupart des méthodes utilisées pour la résolution des problèmes de planification classique STRIPS (espace de recherche discrets) comme les méthodes de recherche heuristique, ne sont pas valables (espace de recherche continue) pour la résolution des problèmes de planification temporelle. Ceci est dû en grande partie à la représentation discrète de l'espace de recherche des solutions. Cependant, pour une résolution non-optimale des problèmes de planification temporelle, on peut transformer dans un premier temps le problème de planification temporelle en problème de planification classique en supprimant les durées des actions. On applique alors un algorithme classique pour le résoudre. Enfin, on réordonne les actions du plan trouvées auxquelles on aura ajouté les durées à l'aide de l'algorithme d'ordonnancement classique : le PERT [Hillier 2001]. Cependant, les solutions obtenues sont souvent de mauvaise qualité.

D'autres approches, afin de tirer profit d'une part des multiples travaux ayant abouti à la définition de plusieurs techniques de résolution des planificateurs classiques, et d'autre part des techniques issues de la recherche opérationnelle comme la programmation par contrainte [Vidal 2004a] ou la programmation linéaire [Wolfman 1999], ont été proposées pour une résolution optimale des problèmes de planification temporelle. Ces techniques bien qu'efficaces, requièrent un grand nombre de variables ce qui limite leur passage à l'échelle.

Les meilleurs planificateurs actuels sont :

– les planificateur LAMA [Richter 2008] et LPG [Gerevini 2003a, Gerevini 2003b] pour les problèmes de planification STRIPS classique,

– le planificateur LAMA pour les problèmes de planification avec coûts,
– et les planificateurs TFD [Eyerich 2009] et LPG pour les problèmes de planification temporels.

Ces planificateurs seront utilisés pour l'évaluation de nos travaux (cf. chapitre 7).

Chapitre 3

Les algorithmes évolutionnaires

Sommaire

Les phénomènes physiques ou biologiques ont été à la source de nombreux algorithmes s'en inspirant plus ou moins librement. Les algorithmes évolutionnaires sont issus d'une métaphore avec l'évolution darwinienne caractérisée par les principes de sélection naturelle et de variation aveugle du matériel génétique.

Née dans les années 60, ce n'est que vers le début des années 90 que cette technique arrive sur le devant de la scène, s'inscrivant un peu, avec les techniques liées à la vie artificielle d'une façon plus générale, comme une voie de renouvellement pour le génie logiciel et l'intelligence artificielle.

On distingue quatre grandes familles historiques d'algorithme et les différences entre elles ont laissé des traces dans le paysage évolutionnaire actuel, en dépit d'une unification de nombreux concepts :

- Algorithmes Génétiques (GA) (ou algorithmes évolutionnaires), proposés par J.Holland [Holland 1975], dès les années 60 dans le Michigan aux USA, et popularisés par son élève D.E. Goldberg [Goldberg 1989] ;
- Stratégies d'Evolution (ES), inventées par I. Rechenberg [Rechenberg 1973] et H.P. Schwefel [Schwefel 1977], dans les années 70 à Berlin ;
- Programmation Evolutionnaire (EP), imaginée par L.J. Fogel et ses coauteurs [Fogel 1966] dans les années 60, et reprise par son fils D.B. Fogel [Fogel 1995] dans les années 90, en Californie, USA ;

– Enfin la Programmation Génétique (GP, *Genetic Programming*), amenée à maturité par J. Koza [Koza 1992, Koza 1994].

Les algorithmes évolutionnaires sont avant tout des méthodes stochastiques d'optimisation globale. Et la souplesse d'utilisation de ces algorithmes pour des fonctions objectif non régulières, à valeurs vectorielles, et définies sur des espaces de recherche non-standards permet leur utilisation pour des problèmes qui sont pour le moment hors d'atteinte des méthodes déterministes plus classiques.

3.1 Terminologie

Soit à optimiser une fonction J à valeur réelle définie sur un espace métrique Ω. Le parallèle avec l'évolution naturelle a entraîné l'apparition d'un vocabulaire spécifique :

– la fonction objectif J est appelée fonction *performance*, ou fonction *d'adaptation* (ou *fitness* en anglais) ;
– les points de l'espace de recherche Ω sont appelés des *individus* ;
– les ensembles d'individus sont appelés des *populations* ;
– on parlera d'une *génération* pour la boucle principale de l'algorithme.

Le temps de l'évolution est supposé discrétisé, et on notera Pop_i la population, de taille fixe p, à la génération i.

3.1.1 Les étapes

La pression de l'*environnement*, qui est simulée à l'aide de la fonction d'adaptation J, et les principes darwiniens de *sélection naturelle* et de *variations aveugles* sont implantés dans l'algorithme de la manière suivante :

– **Initialisation** de la *population* Pop_0 en choisissant p individus dans Ω, généralement par tirage aléatoire avec une probabilité uniforme sur Ω lorsque c'est possible ;
– **Evaluation** des individus de Pop_0 en d'autres termes calcul des valeurs de J pour tous les individus ;
– La génération i construit la population Pop_i à partir de la population Pop_{i-1} :
  – **Sélection** des individus les plus performants *(les plus adaptés)* de Pop_{i-1} au sens de J (les plus adaptés se reproduisent)
  – Application (avec une probabilité donnée) des **opérateurs de variation** aux *parents* (ou individus les plus performants) sélectionnés, ce qui génère de nouveaux individus : les *enfants* ; on parlera de *mutation* pour les opérateurs unaires, et de *croisement* pour les opérateurs binaires (ou n-aires) ; à noter que cette étape est toujours **stochastique**.
  – **Evaluation** des enfants ;
  – **Remplacement** de la population Pop_{i-1} par une nouvelle population créée à partir des enfants et/ou des parents de la population Pop_{i-1} au moyen d'une sélection darwinienne *(les plus adaptés survivent)*.

– L'évolution s'arrête lorsque le niveau souhaité de performance est atteint, ou qu'un nombre fixé de générations s'est écoulé sans améliorer l'individu le plus performant.

Il est important de noter que dans les applications, l'essentiel du coût-calcul de ces algorithmes provient de l'étape d'évaluation : par exemple si les tailles de populations sont de l'ordre de quelques dizaines et le nombre de générations de quelques centaines, ceci donne lieu à plusieurs milliers de calculs de J.

3.1.2 Représentation

La composante principale de l'algorithme est la *représentation* ou choix de l'espace de recherche. Dans de nombreux cas, l'espace de recherche est totalement déterminé par le problème (c'est-à-dire l'espace Ω sur lequel est défini la fonction objectif J).

Mais il est toujours possible de transporter son problème dans un espace habilement choisi (" changement de variables") dans lequel il sera plus facile de définir des opérateurs de variations efficaces. Cet espace est appelé *espace génotypique*. L'espace de recherche initial Ω dans lequel est calculé la performance des individus, prend quant à lui le nom d'*espace phénotypique* .

On peut par conséquent répartir les diverses étapes de l'algorithme en deux groupes : les étapes relatives au **darwinisme artificiel** (sélection et remplacement), qui ne dépendent que des valeurs prises par J, et pas de la représentation choisie ; et les étapes liées à la nature de l'espace de recherche. Ainsi, l'**initialisation** et les **opérateurs de variation** sont spécifiques aux types de génotypes mais ils ne dépendent pas de la fonction objectif J (c'est le principe Darwinien des variations *aveugles*, ou *non dirigées*).

3.1.3 Points-clés

Le terme de *diversité génétique* désigne la variété des génotypes présents dans la population. Elle devient nulle lorsque tous les individus sont identiques, on parle alors de *convergence* de l'algorithme. Il est important de noter que lorsque la diversité génétique devient faible, il y a très peu de chance pour qu'elle augmente à nouveau. Et si cela se produit très tôt, la convergence a lieu en général vers un optimum local : on parle alors de *convergence prématurée*. Il faut donc préserver la diversité génétique sans pour autant empêcher la convergence. Un autre point de vue sur ce problème est celui du *dilemme exploration-exploitation*.

A chaque étape de l'algorithme, il faut effectuer le compromis entre **explorer** l'espace de recherche afin d'éviter de stagner dans un optimum local, et **exploiter** les meilleurs individus obtenus, afin d'atteindre de meilleurs valeurs aux alentours. Une quantité trop élevée d'exploitation entraîne une convergence vers un optimum local et trop d'exploration entraîne la non convergence de l'algorithme.

Le réglage des parts d'exploration et d'exploitation s'effectue en jouant sur les divers paramètres de l'algorithme comme par exemple les probabilités d'application

des opérateurs d'évolution.

Malheureusement, il n'existe pas de règles universelles de réglages et seuls les résultats expérimentaux donnent une idée du comportement des diverses composantes des algorithmes (cf. chapitre 6).

3.2 Les procédures de sélection

La partie darwinienne de l'algorithme comprend les deux étapes de **sélection** et de **remplacement**. Ces étapes sont totalement indépendantes de l'espace de recherche. D'un point de vue technique, la différence essentielle entre l'étape de remplacement et l'étape de sélection est qu'un même individu peut être sélectionné plusieurs fois durant l'étape de sélection (ce qui correspond au fait d'avoir plusieurs enfants) alors que durant l'étape de remplacement, chaque individu est sélectionné une fois (et il survit) ou pas du tout (et il disparait). Remarquons que la procédure de remplacement peut impliquer soit les enfants exclusivement, soit la population précédente dans son ensemble. Néanmoins, les étapes de sélection et remplacement utilisent les mêmes procédures de choix des individus, dont les plus utilisées seront décrites ci-dessous. On distingue deux catégories de procédure de sélection ou de remplacement : les procédures déterministes et les procédures stochastiques.

3.2.1 Les procédures de sélection déterministes

On sélectionne les meilleurs individus au sens de la fonction performance. Ces procédures supposent un tri de l'ensemble de la population. Mais cela ne pose un problème de temps de calcul que pour de très grosses tailles de population.

Les individus les moins performants sont totalement éliminés de la population, et le meilleur individu est toujours sélectionné. On parle alors de sélection *élitiste* .

3.2.2 Les procédures de sélection stochastiques

Il s'agit toujours de favoriser les meilleurs individus, mais ici de manière stochastique ; ce qui laisse une chance aux individus moins performants. Par contre, il peut arriver que le meilleur individu ne soit pas sélectionné, et qu'aucun des enfants n'atteigne une performance aussi bonne que celle du meilleur parent.

3.2.2.1 Le tirage de roulette

C'est la plus célèbre des sélections stochastiques. Supposant un problème de maximisation avec uniquement des performances positives, elle consiste à donner à chaque individu une probabilité d'être sélectionné proportionnellement à sa performance. Une illustration de la roulette est donnée figure 3.1 : on lance la boule dans la roulette, et on choisit l'individu dans le secteur duquel la boule a fini sa course.

Le tirage de roulette présente toutefois de nombreux inconvénients, en particulier reliés à l'échelle de la fonction performance : alors qu'il est théoriquement équivalent d'optimiser J et $\alpha J + \beta$ pour tout $\alpha > 0$, il est clair que le comportement de la

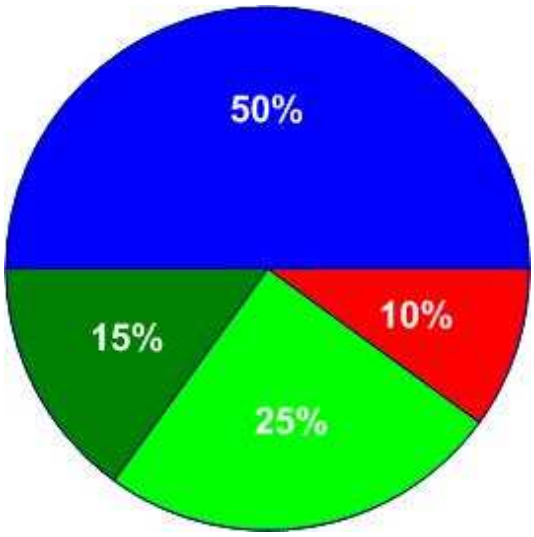

FIG. 3.1 – La roulette - $J(individu_1) = 50,\ J(individu_2) = 15,\ J(individu_3) = 25,\ J(individu_4) = 10$

sélection par roulette va fortement dépendre de α dans ce cas. C'est pourquoi, bien qu'il existe des procédures ajustant les paramètres de α et de β à chaque génération, cette sélection est presque totalement abandonnée aujourd'hui.

3.2.2.2 La sélection par le rang

Elle consiste à faire une sélection en utilisant une roulette dont les secteurs sont proportionnels aux **rangs** des individus (P pour le meilleur, 1 pour le moins bon, pour une population de taille P). La variante linéaire utilise directement le rang, les variantes polynomiales remplaçant ces valeurs par $\frac{(rang)^\alpha}{P}$ avec $\alpha > 0$. Le point essentiel de cette procédure de sélection est que les valeurs de J n'interviennent plus. Seuls comptent aussi les positions relatives des individus entre eux. Optimiser J et $\alpha J + \beta$ sont alors bien et totalement équivalents.

3.2.2.3 La sélection par tournoi

Elle n'utilise que la comparaison entre individus et ne nécessite pas de tri de la population. On distingue deux types de tournoi : le tournoi **déterministe** et le tournoi **stochastique**.

Le premier type possède un paramètre T, taille du tournoi. Pour sélectionner un individu, on tire T individus uniformément dans la population, et on sélectionne le meilleur de ces T individus. Le choix de T permet de faire varier la *pression sélective* en d'autres termes les chances de sélection des plus performants par rapport aux plus faibles.

Pour le second type, $T = 2$ individus sont uniformément choisi dans population et le meilleur des 2 est sélectionné avec une probabilité $t \in [0.5, 1]$.

3.3 Les moteurs d'évolution

On regroupe sous ce nom les ensembles sélection/remplacement, qui ne peuvent être dissociés lors des analyses théoriques du darwinisme au sein des algorithmes évolutionnaires. Un moteur d'évolution est donc la réunion d'une procédure de sélection et d'une procédure de remplacement. Toute combinaison des procédures présentées plus haut (et bien d'autres) est licite. Toutefois, certaines combinaisons sont plus souvent utilisées, que ce soit pour des raisons historiques, théoriques ou expérimentales. Pour cette raison, les noms donnés sont souvent les noms des écoles historiques qui les ont popularisées. Ces schémas sont totalement indépendants de l'espace de recherche et ces écoles historiques travaillaient sur des espaces de recherche bien précis. Voici quelques exemples d'écoles :

3.3.1 Algorithme Génétique Générationnel (GGA)

Ce moteur utilise une sélection stochastique pour sélectionner exactement p parents (certains parents peuvent donc être sélectionnés plusieurs fois, d'autres pas du tout). Ces p parents donnent ensuite p enfants par application des opérateurs de variation (mutation et croisement). Enfin, ces p enfants remplacent purement et simplement les p parents.

3.3.2 Algorithme Génétique Stationnaire (*Steady-state* GA - SSGA)

Dans ce moteur, un individu est sélectionné généralement par tournoi, un second si le croisement doit être appliqué, et l'enfant résultant (après croisement et mutation éventuels) est réinséré dans la population en remplacement d'un "vieil" individu sélectionné par un tournoi inversé, dans lequel le moins performant (ou le plus vieux) "gagne" ... et disparaît.

3.3.3 Stratégies d'Evolution ($(\mu \dot{+} \lambda)$-ES)

Deux moteurs sont regroupés sous ces appellations. Dans les deux cas, l'étape de sélection est un tirage uniforme (on peut dire qu'il n'y a pas de sélection au sens darwinien). A partir d'une population de taille μ (notation historique), λ enfants sont générés par application des opérateurs de variation. L'étape de remplacement est alors totalement déterministe. Dans le schéma (μ, λ)-ES (avec $\lambda > \mu$), les μ meilleurs enfants deviennent les parents de la génération suivante, alors que dans le schéma $(\mu + \lambda)$-ES, ce sont les μ meilleurs des $\mu + \lambda$ parents plus enfants qui survivent.

3.4 Le *framework* évolutionnaire EO

La bibliothèque évolutionnaire EO[1] est un *framework* Open Source écrit en C++, basé sur la librairie STL et des templates, et développé par un consortium de chercheurs (université de Granada, d'Amsterdam, de Berlin, et INRIA - M. Schoenauer). Il offre des représentations standards et des opérateurs génériques de sélection/remplacement, un paramétrage de la ligne de commande et un environnement de conduite de campagne de tests. Disponible depuis 2000, il a été utilisé dans de nombreux contextes académiques et industriels.

Les deux originalités de EO par rapport à d'autres librairies équivalentes sont d'une part l'indépendance par rapport aux grandes familles historiques d'algorithme évolutionnaire, d'autre part la complète indépendance entre l'espace de recherche et le moteur d'évolution.

En ce qui concerne l'espace de recherche, outre les types de bases (chaîne de bits, paramètres réels, arbres de la programation génétique), EO permet d'utiliser n'importe quelle structure de données, offrant même par défaut des opérateurs de variation génériques.

En terme de moteur d'évolution, EO recouvre les moteurs usuels (cf. 3.3), mais permet aussi des expérimentations sur des types originaux.

Grâce à cette généricité, l'implantation d'un algorithme évolutionnaire à l'aide de la bibliothèque EO nécessitera simplement la définition d'un génotype, d'une méthode d'initialisation, des opérateurs de variation et d'une fonction objectif.

3.5 Conclusion

Ce chapitre a présenté de manière générale les algorithmes évolutionnaires. Ce qu'il faut en retenir, c'est d'une part leur robustesse vis-à-vis des optima locaux, et d'autre part leur souplesse d'utilisation, particulièrement en terme de champ d'application. Hormis la définition des individus, de la fonction d'évaluation et des opérateurs de variations, l'efficacité de la méthode nécessite aussi un bon paramétrage de l'algorithme.

Malheureusement il n'existe pas de règles universelles permettant le choix de bons paramètres. Seuls les résultats expérimentaux donnent une idée du comportement des divers composants de l'algorithme.

[1] http ://eodev.sourceforge.net/

Chapitre 4

Les approches évolutionnaires et l'approche Divide-And-Evolve

Sommaire

A cause de la grande complexité combinatoire des problèmes de planification, PSPACE [Bylander 1994a] pour les problèmes de planification temporel simple et EXPSPACE [Rintanen 2007] pour les problèmes de planification temporellement expressifs, les algorithmes évolutionnaires [Eiben 2003] sont de bons candidats pour la mise au point de méthodes généralistes.

4.1 Les algorithmes évolutionnaires et la planification

Depuis les travaux de Koza [Koza 1992] dans les années 90, où il montre sur quelques instances du domaine `Blocks-world`, la possibilité d'utilisation des algorithmes évolutionnaires pour la résolution des problèmes de planification STRIPS classique (qu'il nomme *Genetic Planning*), plusieurs approches s'en inspirant ont été par la suite proposées [Spector 1994, Muslea 1997, Westerberg 2000, Brié 2005].

4.1.1 L'approche *Genetic Planning*

L'approche *Genetic Planning* fait évoluer une population de plans (ou séquences d'actions), initialement générée de manière uniforme. Utilisée que dans le cadre de la planification STRIPS classique, la *fitness* est fonction du nombre d'atomes du

but atteint après application de l'individu (ou du plan) et de la longueur de l'individu. La fitness est calculée en appliquant une procédure qui simulera l'application de l'individu. Si le but[1] n'est pas atteint, une valeur fonction des résultats de la simulation de l'individu sera affectée à la fitness. Les opérateurs de variations quant à eux, pour la création des enfants, tenteront, soit de réordonner les actions d'un individu, soit de faire varier les tailles des individus à l'aide d'un croisement ou d'un ajout (respectivement d'une suppression) d'une ou de plusieurs actions. Trois systèmes se basant sur cette approche, SINERGY [Spector 1994, Muslea 1997], GenPlan [Westerberg 2000] et [Brié 2005], ont ainsi été construits. Cependant, les résultats obtenus par ces systèmes fûrent très inférieurs aux abondants résultats de l'état de l'art.

Toutefois, les algorithmes évolutionnaires ont été utilisés comme une méthode d'apprentissage pour les planificateurs de l'état de l'art. C'est ainsi que tout récemment [Newton 2007] propose une méthode d'apprentissage évolutionnaire permettant de déduire des macros-actions (ou groupes d'actions pouvant être appliquées toutes à la fois dans un état donné) qui seront ensuite proposées aux planificateurs existants afin d'accélérer la résolution des problèmes. Le système L2Plan [Levine 1999, Levine 2009] utilise les algorithmes évolutionnaires pour l'optimisation des politiques de contrôle pour les problèmes de planification non observables[2]. Le système EvoCK [Aler 1998, Aler 2001] quant à lui utilise les heuristiques générées par le système d'apprentissage heuristique HAMLET [Borrajo 1997] afin de construire une population initiale d'heuristiques et les optimise pour le système PRODIGY4.0 [Veloso 1995]. Il est à noter que les résultats du système EvoCK sont meilleurs comparés à ceux des systèmes PRODIGY4.0 et HAMLET.

Il est bien connu que les algorithmes évolutionnaires peuvent rarement résoudre efficacement des problèmes d'optimisation combinatoire par leurs seules capacités sans être hybridés avec les méthodes locales *ad hoc* [Schoenauer 2010]. Les méthodes d'hybridation utilisant les méthodes de recherche opérationnelle pour l'amélioration locale des descendants nés des opérateurs de mutation et de croisement, sont devenues l'objet d'un domaine de recherche [Moscato 2003]. Cependant, la plupart des approches sont basées sur la recherche d'améliorations locales de solutions candidates proposées par le mécanisme de recherche évolutionnaire qui utilise des méthodes locales de résolution devant aborder le problème dans sa totalité.

Dans certains domaines combinatoires, comme par exemple la planification temporelle, cela s'avère tout simplement impossible à partir d'un certain niveau de complexité des problèmes.

Inspiré du paradigme Divide-and-Conquer, un schéma original de collaboration entre un algorithme évolutionnaire et un planificateur exact capable de résoudre de "petits" problèmes a été récemment mis en place par [Schoenauer 2006, Schoenauer 2007] pour ce type de situation. Le problème est découpé arbitrairement en une séquence de sous-problèmes que l'on espère plus simple à résoudre par

[1] Le but est atteint si la conjonctions de tous les atomes du but est vrai dans l'état résultant de l'application séquentielle des actions de l'individu à partir de l'état initial.

[2] Les actions ont des probabilités d'exécution

une méthode locale spécialisée. La solution du problème initial est alors obtenue en concaténant les solutions des différents sous problèmes.

Ce schéma est à la base des travaux de cette thèse. Avant de le présenter en détail, puis de décrire nos contributions, nous présenterons les travaux utilisant également l'idée de décomposition.

4.1.2 La décomposition en planification

Appliquer la métaphore *divide and conquer* pour la résolution des problèmes de planification n'est pas une idée nouvelle dans la communauté planification. [Sebastia 2006] identifie 3 étapes nécessaires pour la résolution d'un problème de planification à l'aide de cette métaphore :

- reformulation du problème initial en un ensemble de sous problèmes (ou le principe de décomposition)
- résolution des différents sous problèmes
- recombinaison des différentes solutions des sous problèmes afin d'obtenir la solution du problème initial.

Proposé par [Sacerdoti. 1974] dans les années 70, les algorithmes hiérarchiques (ou *Factored planning algorithms*), exploitent l'indépendance entre les instances et le domaine, issue de la représentation du problème afin de décomposer le domaine. Tout d'abord, le domaine est représenté par un arbre dont les noeuds sont les sous domaines. Un sous-domaine (ou *factor*) sera un ensemble d'atomes et d'actions n'utilisant que les atomes du sous-domaine. Deux sous-domaines seront voisins (ou connectés) s'ils partagent les mêmes atomes et actions. L'idée, ici, est d'extraire des composantes connexes de l'arbre ainsi construit et, ipso facto réduire les interactions entre les sous-domaines voisins. Une décomposition idéale serait celle qui minimise le nombre d'atomes partagés par les sous-domaines [Amir 2003, Kelareva 2007]. Ensuite, chaque sous-domaine ainsi décomposé sera traité séparément. Les différentes solutions des sous problèmes[3] seront alors réordonnées afin d'obtenir la solution du problème initial.

Les planificateurs hiérarchiques les plus connus [Knoblock 1994, Lansky 1995] sont probablement les meilleurs exemples de tels algorithmes. Toutefois, la décomposition hiérarchique n'est efficace que dans les domaines pour lesquels l'une des composante (ou sous-domaine) peut influencer directement ou indirectement les autres. Lorsque ce n'est pas le cas, ces techniques d'abstraction ne donnent pas (ou très peu) de décomposition, et la recombinaison des solutions des sous problèmes pourrait être dans ce cas plus complexe que la résolution directe du problème initial [Brafman 2006].

Afin de s'affranchir de la dépendance critique de ces approches vis à vis des domaines décomposables, plusieurs approches ont proposé de reporter la décomposition sur l'ensemble des atomes. C'est ainsi que [Koehler 2000, Hoffmann 2004] pro-

[3] Un sous problème de planification $\Pi' = \langle A', O', I, G\rangle$ sera le problème initiale $\Pi = \langle A, O, I, G\rangle$ pour lequel l'ensemble des atomes A et l'ensemble des actions O sont restreints à ceux du sous domaine A' et O'.

posent de décomposer le problème initial en fonction des atomes apparaissant dans toutes les solutions possibles (ou *landmarks*). Cependant, la recherche de ces *landmarks* n'est pas une tâche triviale : complexité PSPACE-difficile [Porteous 2001]. Néanmoins, elle a été l'inspiratrice de nombreuses fonctions heuristiques parmi les plus récentes [Richter 2008, Karpas 2009]. [Chen 2006, Wah 2006] montrent qu'il est possible de décomposer l'état but d'un problème de planification temporelle en fonction des relations mutex entre atomes de l'état but à l'aide d'une représentation du problème de planification initial en problème programmation non linéaire entière mixte (MINLP). Bien que le système résultant *SGPLAN* survole depuis 2004 la partie temporelle de la compétion internationale de planification, plusieurs études théoriques ont permis la mise en évidence des limitations des approches actuelles de résolution des problèmes de planification temporelle. Ainsi, tous les domaines et les problèmes qui ont été utilisés jusqu'à l'avant dernière compétition peuvent toujours être résolus par des planificateurs séquentiels [Cushing 2007]. De plus, comparé au planificateur stochastique LPG [Gerevini 2003a, Gerevini 2003b], ou au planificateur déterministe TFD [Eyerich 2009], SGPLAN produit des solutions de mauvaise qualité.

4.2 Le paradigme *Divide-and-Evolve*

4.2.1 La métaphore du TGV : l'origine

La stratégie *Divide-and-Evolve* est née d'une métaphore sur des problèmes du tracé des chemins du Train à Grande Vitesse français (TGV). Le problème original consiste à calculer le plus court chemin entre deux points d'un paysage géographique avec de fortes contraintes sur la courbure et la pente de la trajectoire. Un algorithme évolutionnaire a été développé [Desquilbet 1992], basé sur le fait que la seule méthode de résolution locale était un algorithme glouton pouvant seulement résoudre des problèmes très simples (c'est à dire sur de courtes distances ou sans obstacles) (voir figure 4.1). L'algorithme évolutionnaire recherche un découpage du chemin global en courts segments consécutifs (voir figure 4.1) tels que l'algorithme local puisse facilement trouver un chemin joignant leurs extrémités. Les individus représentent les ensembles de stations intermédiaires entre la station de départ et le terminus ou station but. La convergence vers une bonne solution est obtenue avec la définition d'opérateurs de variation et de sélection appropriés [Desquilbet 1992].

4.2.2 Application à la planification

Appliqué à la planification, les stations deviennent des états intermédiaires et le chemin est remplacé par une séquence d'actions. Le problème de planification initial est ainsi divisé en une série de sous-problèmes et de *courtes distances* deviennent "*facile*" à résoudre par un planificateur local. L'algorithme évolutionnaire joue alors le rôle d'un oracle désignant des états par lesquels il est préférable de passer.

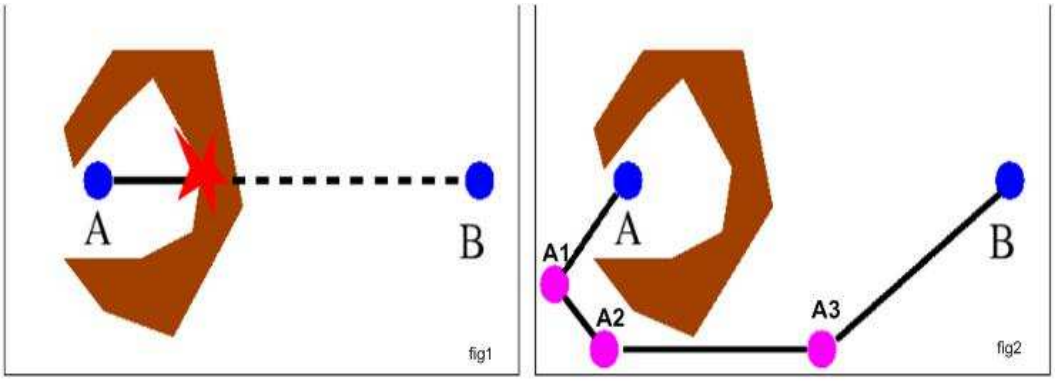

FIG. 4.1 – L'approche *Divide-and-Evolve*. Le problème ici est celui de trouver un chemin de A vers B. La figure fig1 représente le problème vu par un algorithme local. Cet algorithme est incapable de trouver une solution au problème car il se trouve face à un obstacle. Sur la figure fig2, il réussit à surmonter cette difficulté avec l'aide d'un algorithme évolutionnaire. L'algorithme évolutionnaire lui propose des points (A1,A2 et A3) qui rendent les problèmes partiels faciles. Pour l'algorithme local il lui suffit alors de résoudre la séquence des problèmes : trouver les chemins de A vers A1, de A1 vers A2, de A2 vers A3 et de A3 vers B.

4.2.2.1 Représentation

Un individu sera donc une décomposition potentielle (ou une série d'états) en d'autre termes une liste variable d'états. Cependant, construire une série d'états complet entrainerait une explosion combinatoire du nombre d'états possible et par ricochet de l'espace de recherche des séries d'états. De plus, les états but des problèmes de planification sont généralement partiels. Par conséquent, pour réduire l'espace de recherche, il semble donc plus indiqué de construire des séries d'états partiels et de restreindre l'ensemble des atomes servant à les construire. Ainsi, un individu sera une série d'états partiels construits avec un sous-ensemble d'atomes autorisés. Les détails de ce choix d'atomes seront donnés dans le chapitre 5.

4.2.3 L'algorithme Divide-and-Evolve

La figure 4.2 présente les différentes étapes de l'algorithme Divide-and-Evolve. Comme tout algorithme évolutionnaire, Divide-and-Evolve commence par construire de manière stochastique, un ensemble de solutions potentielles au problème posé. Ici, il s'agit d'un ensemble de séries d'états Pop_1 ou population initiale : c'est l'*initialisation*. Cette population est ensuite évaluée.

La boucle principale de Divide-and-Evolve consiste alors à générer une population Pop'_{i+1} à partir de la population Pop_i en appliquant à chaque série d'états de Pop_i des opérateurs de variation (*croisement* et *mutations*) après *sélection* (les séries d'états possédant les meilleures valeurs de fitness sont favorisées). Les séries

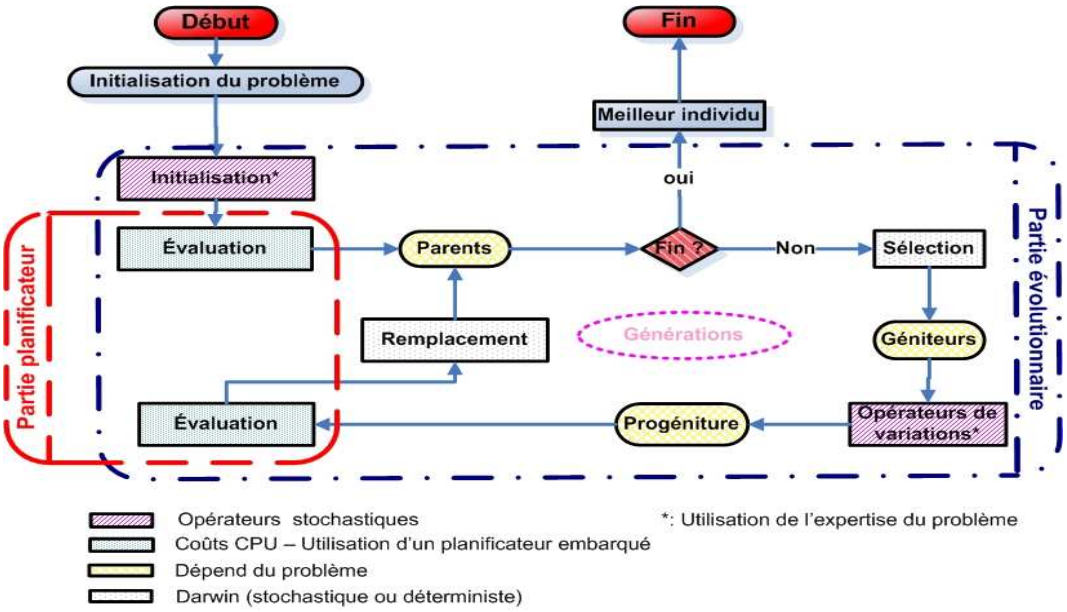

FIG. 4.2 – L'algorithme *Divide-and-Evolve*

d'états de Pop'_{i+1} sont ensuite évaluées et une partie de $Pop_i \bigcup Pop'_{i+1}$ est sélectionnée pour devenir la population Pop_{i+1}. Le processus s'arrête quand on considère que les chances d'amélioration de la fitness dans la population sont faibles – en général après un nombre arbitrairement fixé d'itérations sans amélioration (cf. chapitre 3).

4.3 La fonction d'évaluation

Pour l'évalution d'un individu, Divide-and-Evolve tente de joindre séquentiellement, à partir de l'état initial, les états consécutifs de la série à l'aide d'un planificateur embarqué. Afin de pouvoir résoudre deux états consécutifs de la série d'états, on doit transformer l'état initial en état complet.

4.3.1 Transformation état partiel en état complet

Soit E_c un état complet, s un état partiel, $\Pi = \langle A, O, E_c, s\rangle$ un problème de planification STRIPS et $P = \langle Q_0, Q_1, ..., Q_n\rangle$ une solution de Π

La transformation de s en un état complet (noté $\widehat{s}$) s'effectue en réunissant s avec l'ensemble des atomes présents après application de P auquel on ajoute l'ensemble des atomes de E_c qui n'ont été utilisés par aucune action de P (noté E^-). En d'autres termes

$$\widehat{s} = s \bigcup E^- \bigcup_{i\in[0,n]} [\oplus_{o_j\in Q_i}(add(o_j) - del(o_j))]$$

avec $\oplus$ l'union séquentielle des conséquences d'application des actions des Q_i.

4.3.1.1 Exemple : transformation d'un état partiel en état complet

Soit $\Pi = \langle A, O, I, s\rangle$ un problème de planification STRIPS représentant le jeu du taquin de la figure 1.1 du chapitre 1 ; $s = \{(at\ t\_5\ p\_2\_2), (at\ t\_3\ p\_1\_2)\}$ un état partiel et $P = \langle Q_0 = \{[move\ t\_3\ p\_2\_2\ p\_1\_2]\}, Q_1 = \{[move\ t\_5\ p\_3\_2\ p\_2\_2]\}\rangle$ une solution de Π. Alors la transformation de s en état complet est donnée par la figure 4.3.

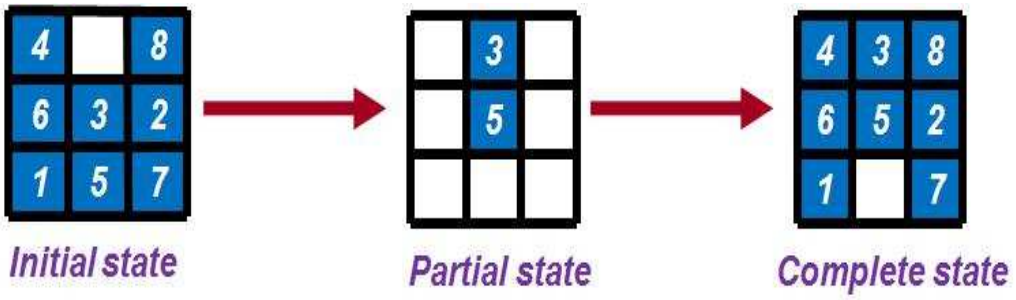

FIG. 4.3 – Jeu du taquin : transformation de l'état partiel $s = \{(at\ t\_5\ p\_2\_2), (at\ t\_3\ p\_1\_2)\}$ en état complet.

4.3.1.2 Fitness

Lors de la résolution d'un problème de planification $\Pi = \langle A, O, I, G\rangle$, la fitness d'une série d'états $S = (s_i)_{i\in[0,n+1]}$ (avec $s_0 = I$ and $s_{n+1} = G$) est calculée en appelant successivement un planificateur embarqué pour la résolution des sous-problèmes de planification $\Pi = \langle A, O, \widehat{s_i}, s_{i+1}\rangle$. Deux situations, dépendant de l'échec ou non du planificateur embarqué lors de la résolution d'un sous-problème, peuvent être envisagées. On parlera d'individu *faisable* si tous ses états sont résolus par le planificateur embarqué et d'individu *infaisable* dans le cas contraire.

La fitness aura pour objectif d'avantager les individus les plus prometteurs au détriment des autres. En d'autres termes il faudrait favoriser un découpage en états aboutissant à une solution. Si aucun individu ne peut trouver de solution, l'individu ayant le plus d'états partiels résolus devra être avantagé en prenant garde aux individus qui auront plusieurs états partiels résolus mais avec un grand nombre d'états intermédiaires inutiles[4]

La détermination de la formule permettant l'évaluation d'un individu est le centre de notre application car c'est elle qui permet le classement de la population et par ricochet la sélection des individus.

Le pseudo-code du calcul de la fitness est donné par l'algorithme 1 [Bibai 2010a]. La boucle principale (lignes 3-12) résout séquentiellement les sous-problèmes de la série d'états à l'aide d'un planificateur embarqué (ligne 5). L'état initial de chaque

[4]Un état sera considérée comme inutile si le passage de l'état précédent à cet état ne nécessite aucune action.

Algorithme 1 Fitness(Ind, planner)

Précondition : I, G, $b_{\max}$, $l_{\max}$

1: $k \leftarrow 0$; $u \leftarrow 0$; $B \leftarrow 0$
2: $\widehat{s} \leftarrow$ I ; $g \leftarrow \{\}$
3: **tant que** $g \neq G$ **faire**
4: $g \leftarrow$ nextGoal(Ind)
5: $(sol_k, b_{\text{done}}) \leftarrow$ planner.Solve$(\widehat{s}, g, b_{\max})$
6: **si** $sol_k = \perp$ **alors**
7: **retour** $(\perp, 10 \cdot l_{\max} \cdot dist(i, G) + \text{length}(Ind) - u)$
8: **sinon si** length$(sol_k) > 0$ **alors** // plan vide ?
9: $u \leftarrow u + 1$ // état utilisé
10: $B \leftarrow B + b_{\text{done}}$ // cumul des efforts
11: $\widehat{s} \leftarrow$ ExecPlan(i, sol_k) // transformation en état complet
12: $k \leftarrow k + 1$ // compteur d'états résolus
13: $(Sol, Q) \leftarrow$ Compress$((sol_j)_{0 \leq j \leq k})$
14: **retour** $(Sol, Q + \frac{length(Ind)-u+1}{Q} + \frac{B}{l_{\max} \cdot b_{\max}})$

sous-problème est obtenu par application du plan (ou de la solution) résultant de la résolution du sous problème le précédant (ligne 11). Le dernier argument du planificateur embarqué $b_{\max}$ (ligne 5) est une borne (ou *mesure de difficulté*) dépendant du planificateur choisi. Elle a pour but de restreindre l'espace de recherche des séries d'états, aux sous-problèmes plus *faciles* à résoudre que le problème de planification initiale Π. En effet, comme il n'existe pas à notre connaissance d'indicateur définissant la difficulté d'un problème de planification donné, il nous a semblé pertinent de contraindre le planificateur embarqué en fonction de ses caractéristiques propres. Ce sera par exemple, pour le planificateur CPT [Vidal 2004a, Vidal 2006] ce sera le *nombre de backtracks* maximal autorisé (cf. section 2.4.6), pour le planificateur YAHSP[Vidal 2004b], le *nombre de noeuds* maximal autorisé à étendre lors de la résolution d'un problème de planification (cf. section 2.3.1) ou encore pour les deux planificateurs *la durée de résolution* maximale autorisée.

Une fois un sous-problème résolu, le planificateur embarqué retourne la solution sol_k et la valeur de la contrainte (ou de la *mesure de difficulté*) effectivement utilisée pour sa résolution. Il est à noter que si le sous-problème n'est pas résolu, la valeur de cette contrainte sera égale à $b_{\max}$ et aucun plan ne sera retourné.

Dans ce cas, l'individu est dit *infaisable* en d'autre termes il existe un état *difficille* dans la série d'états représentant l'individu. Pour ces individus, la fitness est calculée d'après la formule suivante :

$$10 \cdot l_{\max} \cdot dist(i, G) + \text{length}(Ind) - u \tag{4.1}$$

avec $l_{\max}$ le nombre maximal d'états autorisé pour un individu, Ind la série d'états (ou l'individu) et length(Ind) son nombre d'états, u le nombre d'états utiles, et $dist(i, G) = nbTotalGoal - nbGoalCurrent$ la distance syntaxique entre l'état complet courant et l'état final. Avec $nbTotalGoal$ le nombre d'atomes du but et $nbGoalCurrent$ le nombre d'atomes du but qui sont vrais dans l'état courant.

Le but de cette formule est de favoriser les individus *infaisables* pour lesquels les derniers états complets atteints sont proches d'un état contenant G (d'où le coefficient $10 \cdot l_{\max}$ du terme $10 \cdot l_{\max} \cdot dist(i, G)$). Pour deux individus atteignant deux états équidistants de G, celui qui aura le plus grand nombre d'états utiles (ligne 8 - le terme length$(Ind) - u$) sera favorisé. Si les dates de création des états sont connues et que les états d'un individu donné sont ordonnés en fonction de ces dates, alors la formule 4.1 peut être simplifiée et remplacée par :

$$l_{\max} - u \tag{4.2}$$

En effet, plus un état aura une date de création élevée (ou plus d'états utile u) plus il sera proche de l'état but.

Lorsque qu'un individu est *faisable* (tous les états de la série le composant ont été résolus), la concaténation des solutions partielles est une solution du problème initial Π. Cependant, il peut être possible, par exemple pour les problèmes de planification temporelle, de réordonner ces solutions partielles afin d'améliorer la qualité globale de la solution du problème initial Π (cf. section 4.3.2). On obtient alors une qualité Q du plan global.

La fitness des individus *faisables* est donnée par la formule :

$$Q + \frac{length(Ind) - u + 1}{Q} + \frac{B}{l_{\max} \cdot b_{\max}} \tag{4.3}$$

avec Ind l'individu à évaluer et $length(Ind)$ sa taille (ou le nombre d'états de la série), u le nombre d'états utiles, B la somme cumulée des contraintes effectivement utilisées lors de la résolution des sous problèmes de Ind, $l_{\max}$ le nombre maximal d'états autorisés pour un individu, et $b_{\max}$ la contrainte maximale autorisée pour la résolution d'un sous problème par le planificateur embarqué.

Cette formule est en fait artificielle, et est une manière de prendre en compte des critères hierarchiques : l'idée est de favoriser tout d'abord les individus ayant la meilleure qualité globale quelle que soit sa taille (le terme Q). Ensuite, pour des individus de même qualité globale, de favoriser ceux qui auront le plus grand nombre d'états utiles (le terme $\frac{length(Ind)-u+1}{Q}$). Enfin, pour des individus de même qualité globale et de même nombre d'états utiles, de favoriser ceux qui auront été faciles à résoudre par le planificateur local (le terme $\frac{B}{l_{\max} \cdot b_{\max}}$)).

Lors de la comparaison entre deux individus quelconques, l'individu *faisable* sera toujours préféré à l'individu *infaisable* quelles que soit les valeurs de leurs fitness respectives. Deux individus *faisables* seront comparés en fonction de la valeur de fitness retournée à la ligne 14 et deux individus *infaisables* seront comparés en fonction de la valeur retournée à la ligne 7.

4.3.1.3 Comportement pratique

La figure 4.4 présente un exemple du comportement en pratique de la fonction d'évaluation sur l'instance 11 du problème `openstacks-ipc6-satisficing-track`. On remarque qu'avant la génération numéro 8, l'équation 4.1 guide effectivement les

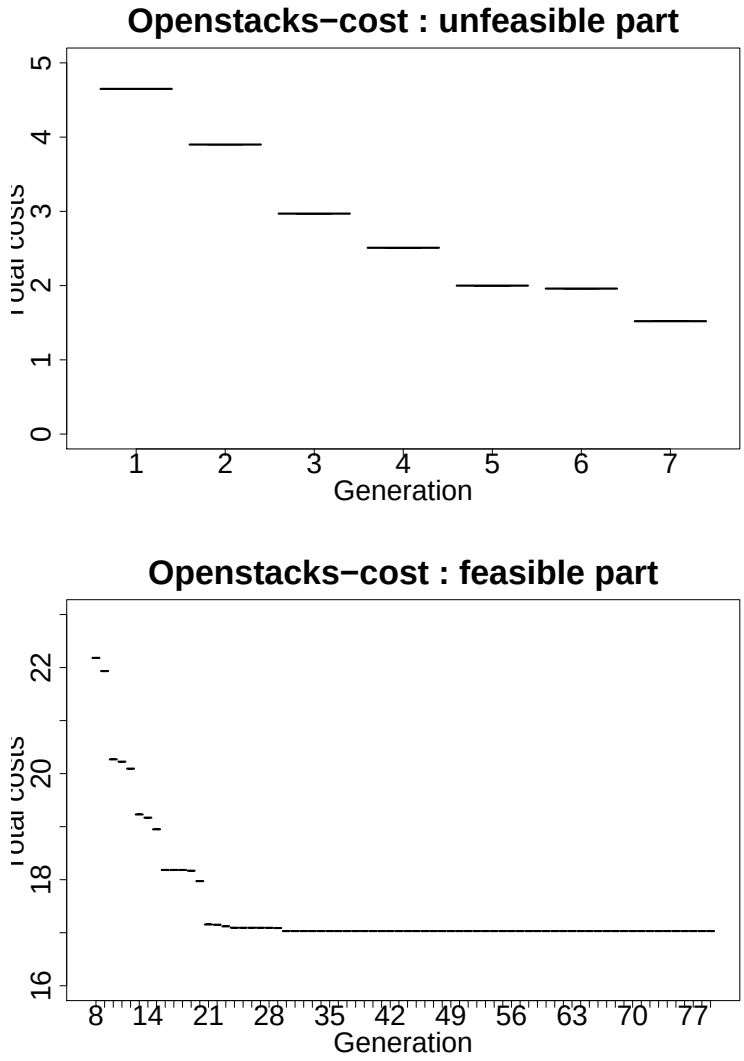

FIG. 4.4 – Evolution de la fitness des individus infaisable vers les individus faisables dans Divide-and-Evolve.

individus non faisables vers ceux faisable (cf. figure 4.4-haut). Lorsqu'un individu faisable apparait dans la population -à partir de la génération numéro 8 (cf. figure 4.4-bas)-, la formule 4.3 prend le relai et optimise la qualité du plan.

La figure 4.5 présente deux exemples typiques du comportement de la fitness des individus faisables observés durant nos expérimentations (cf. chapitre 7). Il est à noter que la qualité des meilleurs individus augmente graduellement en fonction des évaluations (ou des générations).

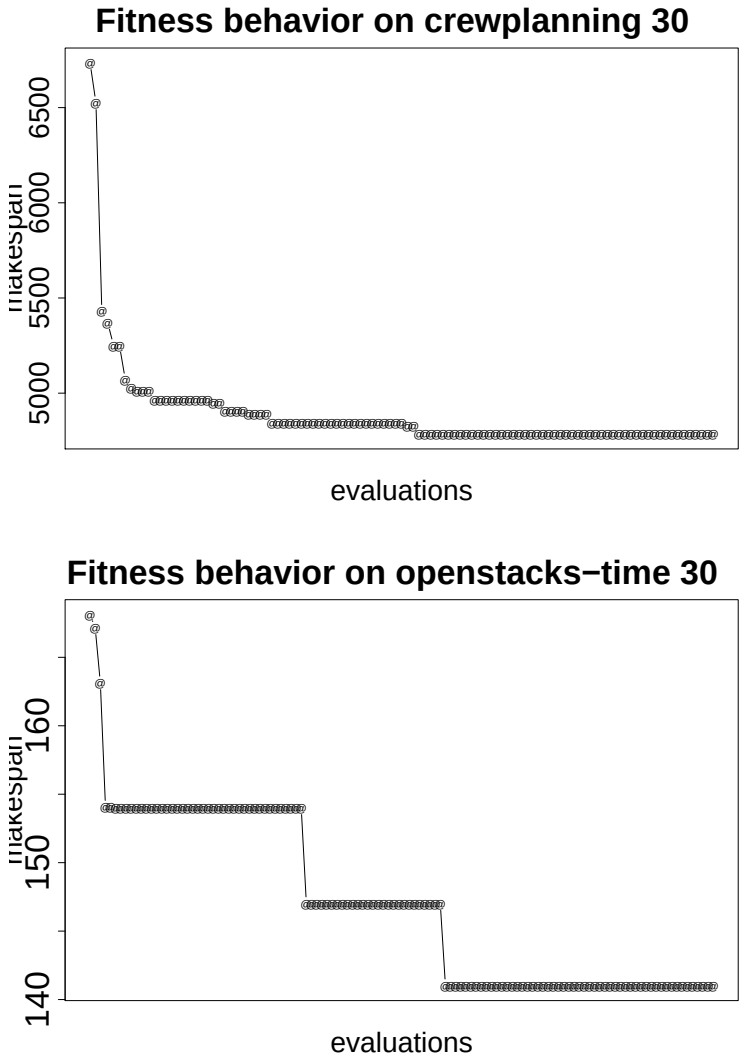

FIG. 4.5 – Exemples typiques de l'évolution de la fitness des individus faisables en fonction des évaluations (ou générations) dans Divide-and-Evolve.

4.3.2 Recomposition des solutions partielles

Comme nous l'avons vu ci-dessus, lorsque tous les états de la série définissant un individu ont été totalement résolus, la concaténation des solutions partielles est une solution du problème initial Π. Cependant, ce plan peut être facilement optimisé en utilisant l'algorithme PRF (cf. [Regnier 1991]), révisé et formalisé par [Bäckström 1998] (page 119) afin de trouver un réordonnancement optimal de ce plan en nombre de niveaux du plan (c'est-à-dire en nombre d'ensembles d'actions indépendants). Le réordonnancement est composé de deux parties. Tout d'abord, un graphe représentant les contraintes entre les actions du plan concaténé est construit.

Algorithme 2 tri-topologique($G(N, C)$)

Précondition : $G(N, C)$: Graphe orienté. N ensemble des noeuds associés aux actions
Précondition : C : ensemble des arcs représentant les contraintes entre actions
1: $sans\_pre \leftarrow \{\}$
2: $Plan \leftarrow \langle\rangle$
3: **répéter**
4: $sans\_pre \leftarrow \{n_1 \in N | \forall n_2 \in N, (n_1, n_2) \notin C\}$
5: **si** $sans\_pre = \{\}$ **alors**
6: **retour** échec // il existe un circuit dans $G(N, C)$
7: $Plan \leftarrow Plan \bigoplus sans\_pre$ // $\bigoplus$ désigne l'opérateur concaténation
8: $N \leftarrow N - sans\_pre$
9: $C \leftarrow C - \{(n_1, n_2) \in C | n_1 \in sans\_pre\}$
10: **jusqu'à** $N = \{\}$
11: **retour** $Plan$

Les contraintes entre les actions sont les relations d'ordre et d'indépendance entre actions. Ensuite on effectue un tri-topologique du graphe obtenu dans le but de trouver un plan minimal en nombre de niveaux.

L'algorithme 2 présente le tri-topologique d'un graphe $G(N, C)$ construit à partir de la concaténation des solutions partielles d'une séquence d'états. le premier niveau est composé des actions indépendantes de $G(N, C)$ (voir ligne 4). Pour la construction des niveaux suivants, on supprime de $G(N, C)$ toutes les actions (ou noeuds) du niveau précédent ainsi que tous les arcs les liant (voir lignes 8 et 9). Enfin le niveau sera composé des actions indépendantes du graphe $G(N, C)$ ainsi modifié.

4.4 Conclusion

Dans le but de transférer l'efficacité des algorithmes évolutionnaires pour la résolution des problèmes combinatoires aux problèmes de planification, plusieurs approches basées sur une représentation directe ont été proposées. Cependant, les résultats obtenus par ces méthodes étaient très inférieurs à ceux de l'état de l'art. Tout récement, l'approche hybride Divide-and-Evolve a été introduite dans le but de faire changer la situation.

Comme tout algorithme évolutionnaire hybride, Divide-and-Evolve nécessite des opérateurs d'initialisation et de variations spécifiques qui influenceront le comportement global de l'algorithme. Comment seront construits ces opérateurs ? Ce sera l'objet du prochain chapitre.

Au début de nos travaux, l'applicabilité et l'efficacité de l'approche Divide-and-Evolve n'ont été démontrées que sur trois instances du domaine Zeno[5] (compétition de planification 2000 - voir annexes) avec l'utilisation des paramètres pour la plupart définis manuellement. De plus, les meilleures solutions trouvées par l'algorithme ainsi développé étaient toujours obtenues dans les toutes premières étapes des différentes

[5] http ://planning.cis.strath.ac.uk/competition/domains.html

exécutions.

Chapitre 5

Construction des séries d'états

Sommaire

Nous nous intéresserons dans cette partie à la construction, dans l'approche Divide-and-Evolve, des séquences d'états que l'on espère faciles à résoudre. C'est la partie majeure de l'algorithme car elle aura un impact sur l'exploration et l'exploitation de l'espace de recherche. [Schoenauer 2006, Schoenauer 2007] ont proposé, dans l'article d'introduction à la méthode Divide-and-Evolve, une approche aveugle de construction des séquences d'états nécessitant un choix expert des atomes permettant la construction des séquences d'états. Cependant, nos travaux ont montré empiriquement [Bibai 2009c] que cela ne constituait pas obligatoirement le meilleur. C'est ainsi que nous avons introduit [Bibai 2009b, Bibai 2009c, Bibai 2010b] une nouvelle méthode dite de *construction orientée dates* [Bibai 2010a] afin de s'affranchir de certains paramètres critiques de l'approche de construction historique. Ces travaux ont également permis le développement de deux systèmes reposant sur ces deux approches [Bibai 2008a, Bibai 2010a]. Le premier [Bibai 2008a], qui implémente l'approche de construction aveugle, a participé à la dernière compétition

internationale de planification devenant ainsi le tout premier planificateur évolutionnaire à y participer depuis sa création.

Ce chapitre sera divisé en 5 parties. Nous commencerons par quelques définitions suivies d'une présentation succincte de l'approche Divide-and-Evolve. Ensuite, nous donnerons en détail la description de chacune des deux approches de construction de séries d'états et enfin nous conclurons.

5.1 Définitions

5.1.1 Date au plus tôt d'un atome, date au plus tôt d'un état

5.1.1.1 Date d'une action

La date d'une action o (noté $date(o)$) est l'instant de terminaison de o (ou la position de o) dans un plan donné.

5.1.1.2 Date d'un ensemble d'actions

La date d'un ensemble d'actions Q (noté $DATE(Q)$) est le maximum des dates des actions de Q. En d'autres termes

$$DATE(Q) = max_{o \in Q}(date(o))$$

5.1.1.3 Date au plus tôt d'un atome

Soit $\Pi = \langle A, O, I, G \rangle$ un problème de planification STRIPS et $P = \langle Q_0, Q_1, ..., Q_j, ..., Q_n \rangle$ une solution de Π ; soit a un atome de A. La date au plus tôt d'apparition de l'atome a (noté $T(a)$) est la date minimale de toutes actions o des Q_i *avec* $i \in [0, n]$ ajoutant a. En d'autres termes, soit $j \in [0, n]$ le plus petit indice tel que $\exists o \in Q_j \ ; \ a \in add(o)$. Soit Q_j^a le sous-ensemble Q_j contenant les actions qui ajoutent a, alors

$$T(a) = min_{o \in Q_j^a}(date(o)) + \sum_{0 \leq i < j} DATE(Q_i)$$

Comme en pratique la solution d'un problème de planification est inconnue, nous utiliserons une estimation de $T(a)$ calculé à l'aide d'une fonction heuristique (cf. chapitre 2 section 2.3.3).

5.1.1.4 Date au plus tôt d'un état

Soit $\Pi = \langle A, O, I, G \rangle$ un problème de planification STRIPS et $s \subseteq A$ un état partiel. La date au plus tôt de s (noté $\triangle\ (s)$) est le maximum des dates au plus tôt des atomes de s. En d'autres termes,

$$\triangle\ (s) = max_{a \in s} T(a)$$

5.1.2 Atomes en permanence mutuellement exclusifs, voisinage temporel

5.1.2.1 Atomes en permanence mutuellement exclusifs

Soit deux atomes a et b de A. On dira que a et b sont en permanence mutuellement exclusifs (noté $mutex(a, b)$) s'il ne peut pas exister d'état contenant à la fois a et b. On notera par $\neg mutex(a, b)$ la négation de cette propriété c'est-à-dire qu'il peut exister un état contenant à la fois a et b.

On notera $M(a) = \{b \in A | mutex(a, b)\}$, la restriction de A aux atomes en permanence mutuellement exclusifs avec l'atome a.

5.1.2.2 Etat partiel admissible

$s \subseteq A$ *état partiel admissible* $\Leftrightarrow \forall a, b \in s, \neg mutex(a, b)$ et $card_s < card_{max}$ (cf. section 2.1.2.2)

Un état partiel admissible n'est pas équivalent à un état partiel car il interdit les états cohérents impossibles (cf. figure 5.1).

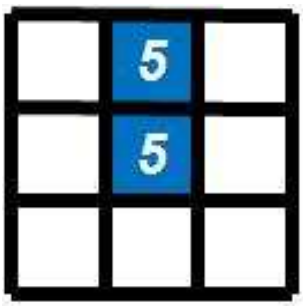

FIG. 5.1 – Etat partiel cohérent mais impossible dans le jeu du taquin à 8 carreaux.

Dans la suite, un état partiel désignera un état partiel admissible.

5.1.2.3 Voisinage temporel

Soit $T = \{t_1, \ldots, t_{i-r}, \ldots, t_i, \ldots, t_{i+r}, \ldots, t_n\}$ un ensemble ordonné de n dates, soit r un entier positif tel que $r < n$, soit $t \in T$ une date de T et i la position de la date t dans T ($t = t_i$). Soit t_{i-r} la date de T se trouvant à la position $i - r$, et soit t_{i+r} la date de T se trouvant à la position $i + r$. Le voisinage temporel de t de rayon r, noté $N_t(t, r)$, est l'ensemble composé d'au plus $2 * r + 1$ dates défini par :

$$N_t(t = t_i, r) = \{d \in T \mid t_{i-r} \leq d \leq t_{i+r}\}$$

avec $t_{i-r} = t_1$ si $i - r < 1$ et $t_{i+r} = t_n$ si $i + r > n$.

5.2 Construction des séries d'états

Comme indiqué au chapitre 4, *Divide-and-Evolve* (*D&E*) est une approche stochastique pour la résolution des problèmes de planification STRIPS, indépendante du domaine. *D&E* tente de diminuer la complexité d'un problème de planification STRIPS $\Pi = \langle A, O, I, G\rangle$ en générant une suite ordonnée d'états partiels de taille variable $s_0 = I, s_1, \ldots, s_{n+1} = G$, et en résolvant successivement les sous-problèmes de planification temporelle $\Pi_i = \langle A, O, \widehat{s_i}, s_{i+1}\rangle$, dont on espère qu'ils sont plus simples à résoudre que le problème initial Π. Cette suite $s_1, ..., s_n$ est générée et optimisée par un algorithme évolutionnaire (cf. chapitre 3), chaque sous-problème étant résolu par un planificateur embarqué.

Proposition : existence d'une décomposition

Pour tout problème de planification STRIPS $\Pi = \langle A, O, I, G\rangle$ admettant une solution $P = \langle Q_0, Q_1, ..., Q_n\rangle$, il existe une décomposition de Π en série d'états conduisant à P.

5.2.0.4 Preuve

Il suffit de construire la série d'états $Ind = (s_i)_{i\in[0,n+1]}$ de la manière suivante

$$s_i = \cup_{o_j\in Q_i}(add(o_j)) - \cup_{o_j\in Q_i}(del(o_j)) \; avec \; i \in [0, n+1]$$

Remarque : Sans pour autant hypothéquer les chances de succès, les s_i sont alors complets.

5.2.1 Plan d'une série d'états

Soit $\Pi = \langle A, O, I, G\rangle$ un problème de planification STRIPS ; Soit $\Pi_1 = \langle A, O, I, s_1\rangle$; $\Pi_i = \langle A, O, \widehat{s_i}, s_{i+1}\rangle$ *avec* $i \in [1, n]$; et $\Pi_{n+1} = \langle A, O, \widehat{s_n}, G\rangle$, $n+1$ sous-problèmes de planification STRIPS du problème Π, et $P_1, P_2, ..., P_{n+1}$ les $n+1$ solutions respectives des Π_j. Alors une solution P du problème Π est la concaténation séquentielle des plans P_j *avec* $j \in [1, n+1]$.

La qualité d'une série d'états $s_0 = I, s_1, \ldots, s_{n+1} = G$ est *évaluée*, en appliquant le planificateur embarqué successivement à chaque sous-problème $\Pi_j, j = 0, \ldots, n+1$; si l'un des problèmes Π_j n'est pas résolu par le planificateur embarqué, l'évaluation s'arrête et la série d'états reçoit une mauvaise évaluation, sinon l'ensemble des solutions partielles est ensuite réordonné, et l'évaluation de la série d'états est la qualité totale du plan réordonné (cf. section 4.3).

5.2.2 Approches de construction des séries d'états

La construction des séries d'états (ou solutions candidates) est un des points critiques de l'algorithme *D&E* car elle doit permettre de guider la recherche des

solutions du problème initial (Π) grâce à la construction des séries de sous-problèmes "faciles" à résoudre par un planificateur embarqué.

Deux aspects principaux guideront la conception de l'algorithme : essayer de respecter la cohérence des états s_i, et restreindre au maximum la taille de l'espace de recherche en utilisant des connaissances a priori que l'on peut déduire du problème. Deux points de vue vont être envisagés, différents par le degré de prise en compte des relations entre atomes qui constituent un état donné. Ainsi, dans la **construction aveugle**, nous nous intéresserons à la construction des séries d'états $s_0 = I, s_1, \ldots, s_{n+1} = G$ grâce à des choix uniformes des atomes dans l'ensemble A tout en garantissant la cohérence 2 à 2 des atomes dans chaque s_i. La **construction orientée**, quant à elle, utilisera en plus une estimation des dates au plus tôt d'apparition d'un atome dans toute solution du problème lors de la construction de la série d'états $s_0 = I, s_1, \ldots, s_{n+1} = G$.

Nous allons dans la suite nous concentrer sur les étapes d'**initialisation**, de **mutation** et de **croisement**. Dans les deux approches de construction des séries d'états, comme les séquences d'états sont des listes variables d'états partiels qui sont aussi des ensembles variables d'atomes, les opérateurs de mutations procéderont :

- soit en supprimant un atome d'un état de la séquence choisi uniformément,
- soit en supprimant un état de la série d'états choisi uniformément,
- soit en ajoutant un atome d'un état de la séquence,
- soit en ajoutant un état dans la série d'états.

Dans ce qui suit, nous noterons par $\mathbb{U}(X)$ le tirage uniforme d'un élément de l'ensemble X, et par $dernier(Ind)$ la position du dernier état d'une séquence d'états $Ind = (s_1, \ldots, s_{dernier(Ind)}, \ldots, s_n)$ résolu par le planificateur embarqué choisi. Il est à signaler que si tous les états sont résolus alors $dernier(Ind)$ sera la taille de l'individu plus un ($s_{dernier(Ind)} = s_{n+1} = G$).

5.3 Construction aveugle

Nous nous intéresserons, dans cette partie, à la construction des séries d'états s_i grâce à des choix uniformes des atomes dans l'ensemble des atomes de base A, tout en garantissant la cohérence 2 à 2 des atomes lors des créations ou modifications d'états. Afin de réduire l'espace de recherche des séquences d'états, nous allons essayer de restreindre l'ensemble des atomes A par le choix des prédicats autorisés pour la construction des états. Cette méthode de construction des séries d'états, dont l'étude de faisabilité a été montrée dans [Schoenauer 2006, Schoenauer 2007], et sa mise en oeuvre pratique dans [Bibai 2008b], a été utilisée pour l'implantation du système DAE-1 [Bibai 2008a] qui a participé à la dernière compétition internationale de planification.

5.3.1 Choix des atomes autorisés

Le choix des prédicats (ou des atomes autorisés) pour la construction des états doit permettre non seulement de réduire l'espace de recherche Ω mais aussi d'accé-

lérer la recherche des solutions du problème Π, tout en garantissant, si possible, la meilleure qualité des séquences d'états construites.

La première approche naturelle qui vient immédiatement à l'esprit est le choix des prédicats ayant permis la construction de l'état partiel final G. Ainsi, la recherche d'une solution consistera alors à un partitionnement de G en plusieurs sous-ensembles d'atomes de G. Cependant, si l'état final G ne contient qu'un seul atome caractérisant par exemple un ensemble de faits, considérer l'ensemble des prédicats de G ne sera pas une bonne option. Dans tous les cas, quelque soit la séquence d'états considérée, le dernier état de cette séquence sera toujours l'état G. Comment donc choisir dans ce cas certains atomes de A sans connaissance a priori ?

Nous avons proposé une règle empirique [Bibai 2008a] permettant de restreindre l'ensemble A des atomes de base aux atomes construits à partir d'un nombre de prédicats fixés. Cette règle (voir figure 5.2) se base sur les proportions d'atomes construits à partir d'un prédicat donné afin de déduire un sous-ensemble de prédicats autorisés pour la construction des séquences d'états. Le choix de n'utiliser que les prédicats d'arité un et deux a été influencé par les opérateurs de variations. En effet, lors de la conception des opérateurs de variation, nous avons été amenés à définir des opérateurs de déplacement dans l'espace de recherche. Dans le cas de la construction aveugle, pour un atome a donné, il nous a semblé logique que la meilleure stratégie de déplacement était la variation d'un des arguments du prédicat utilisé pour l'instanciation de a. Dans le but de réduire l'espace de recherche des séries d'états, le choix de l'ensemble des prédicats autorisés a été restreint à ceux d'arité un et deux.

5.3.2 Dimensionnement

Il est a priori très difficile de décider de la taille (ou du nombre d'atomes) d'un état partiel, ainsi que de celle d'une séquence d'états partiels.Dans le but de réduire le plus possible le nombre de paramètres à régler, nous avons choisi de fixer empiriquement :

- la taille maximale d'un état à la médiane des instances de prédicats choisis pour la construction des séquences d'états et
- celle des séquences d'états à trois fois le nombre d'atomes de l'état final G.

Néanmoins, l'utilisateur peut, s'il le souhaite, modifier ces bornes.

5.3.3 Initialisation

L'algorithme 3 présente l'implantation de l'initialisation aveugle. Elle consiste en un choix uniforme de la taille N de la séquence d'états initiale dans $[1, \#goal]$[1], où $\#goal$ désigne le nombre d'atomes de l'état final. Si l'état final contient un seul atome, ce qui est rarement le cas, on peut remarquer dans ce cas que tous les

[1]Les opérateurs de variation peuvent augmenter la taille des séquences d'états, dans la limite des bornes (cf. section 5.3.2).

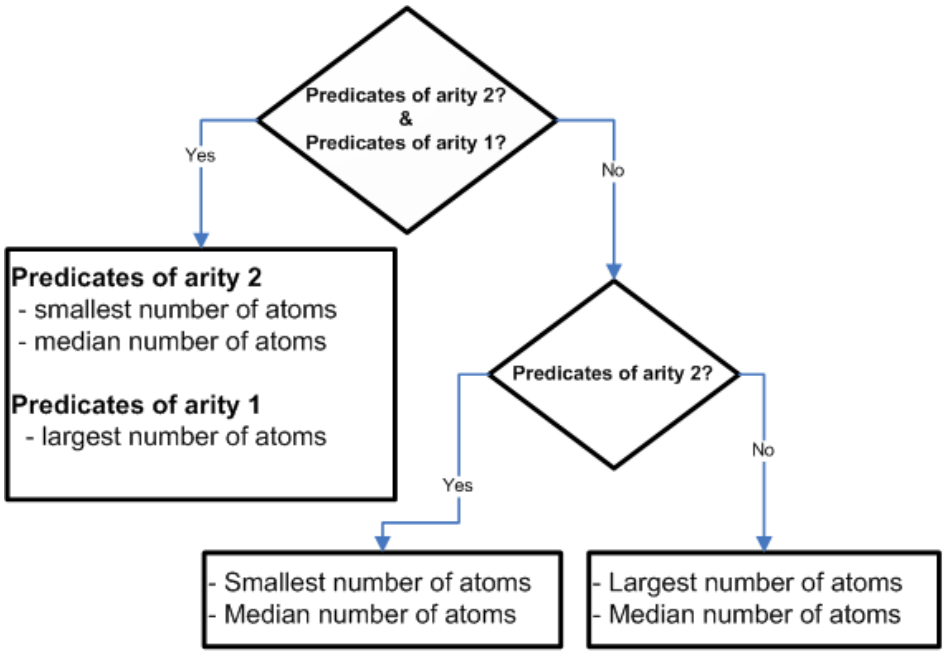

FIG. 5.2 – Choix des prédicats. Si le domaine contient à la fois des prédicats d'arité 1 et 2, l'ensemble des prédicats autorisés seront ceux ayant la plus petite et la médiane des proportions d'atomes dans A pour les prédicats d'arité 2, et le prédicat d'arité 1 ayant la plus grande proportion d'atomes dans A. S'il existe uniquement des prédicats d'arité 2, ceux retenus seront ceux ayant la plus petite et la médiane des proportions d'atomes dans A. Et s'il existe uniquement des prédicats d'arité 1, ceux retenus seront ceux ayant la plus grande et la médiane des proportions d'atomes dans A.

Algorithme 3 InitialisationAveugle(Predicats,$\#goal, \#atomes$)

```
1: Ind ← {}
2: N ← 𝕌(#goal)                                   // Taille de l'individu
3: Å ← {a ∈ A | ∃p ∈ Predicats, a instance de p}
4: pour i = 1 to N faire
5:    s ← {}                                      // début de construction d'un état
6:    n ← 𝕌(#atomes)                              // Taille de l'état
7:    tant que n ≠ 0 ∧ Å ≠ {} faire
8:       a ← 𝕌(Å)                                 // choix uniforme d'un atome
9:       s ← s ∪ {a}                              // l'ajouter à l'état s
10:      Å ← Å \ ({a} ∪ M(a))                     // retirer tous les mutex
11:      n ← n − 1
12:   Ind ← Ind ∪ {s}
13: retour Ind
```

Algorithme 4 Croisement(Ind_1,Ind_2)

```
1: s_{i_1} ← 𝕌(Ind_1)                             // Ind_1 = (s_i)_{1≤i≤n}
2: t_{i_2} ← 𝕌(Ind_2)                             // Ind_2 = (t_i)_{1≤i≤m}
3: Ind'_1 ← ⟨s_1, ..., s_{i_1}, t_{i_2+1}, ..., t_m ⟩   // premiere séquence d'états
4: Ind'_2 ← ⟨t_1, ..., t_{i_2}, s_{i_1+1}, ..., s_n ⟩   // seconde séquence d'états
5: retour Ind'_1,Ind'_2
```

individus seront de taille équivalente. Ensuite, les N états de la séquence seront construits séquentiellement. Un état sera construit de la manière suivante :

- tout d'abord, le nombre d'atomes n de l'état (ou la taille de l'état) sera uniformément choisi entre $[1, \#atomes]$,
- ensuite, un atome a sera choisi uniformément dans l'ensemble des atomes autorisés (ligne 3) et sera ajouté à l'état s'il ne contient pas d'atomes contradictoire (ou mutex) avec a.
- enfin, cette opération sera répétée jusqu'à ce que la taille de l'état soit atteinte (lignes 7 à 11).

5.3.4 Croisement

L'algorithme 4 présente l'implantation du croisement. Afin de conserver au mieux l'ordre entre les séquences d'états apprises au cours de l'évolution, nous avons choisi un croisement à 1 point pour les génotypes variables [Schoenauer 2006]. Soient $Ind1 = (s_j)_{j \leq n}$ et $Ind2 = (t_k)_{k \leq m}$ deux séquences d'états (parents). Soient Ind'_1 et Ind'_2 les séquences d'états obtenues après application de l'opérateur de croisement (enfants). Le croisement en un point de $Ind1$ et $Ind2$ consistera en un choix uniforme de deux indices i_1 et i_2 (lignes 1 et 2) de $Ind1$ et $Ind2$, puis les séquences d'états $Ind1'$ et $Ind2'$ seront construites en croisant les deux extrêmités des sous-séquences $s_1, ..., s_{i_1}$; $s_{i_1+1}, ..., s_n$ et $t_1, ..., t_{i_2}$; $t_{i_2+1}, ..., t_m$ (lignes 3 et 4).

Algorithme 5 AjoutEtatAveugle($Ind = (s_i)_{i \leq n}$)

1: $j \leftarrow \mathbb{U}([1, \min(n,\text{dernier(Ind)})])$ // choix d'un indice
2: $A_m \leftarrow \{a \in s_j \mid \exists b \in s_{j+1}, \mathit{mutex}(a,b)\} \cup \{a \in s_{j+1} \mid \exists b \in s_j, \mathit{mutex}(a,b)\}$
3: $A_c \leftarrow \{a \in A \mid a \in s_j \cap s_{j+1}\}$
4: $A_r \leftarrow (s_j \cup s_{j+1}) \setminus (A_m \cup A_c)$
5: $s \leftarrow A_c$ // debut de construction de l'état
6: **tant que** $A_m \neq \{\}$ **faire**
7: $\quad a \leftarrow \mathbb{U}(A_m)$
8: $\quad s \leftarrow s \cup \{a\}$
9: $\quad A_m \leftarrow A_m \setminus (\{a\} \cup M(a))$
10: $n \leftarrow \mathbb{U}(\#A_r)$ // nombre d'atomes supplémentaire
11: **tant que** $n \neq 0 \wedge A_r \neq \{\}$ **faire**
12: $\quad a \leftarrow \mathbb{U}(A_r)$
13: $\quad$ **si** $\forall b \; in \; s, \; \neg \mathit{mutex}(a,b)$ **alors**
14: $\quad\quad s \leftarrow s \cup \{a\}$
15: $\quad\quad n \leftarrow n - 1$
16: $\quad A_r \leftarrow A_r \setminus \{a\}$
17: $\mathit{insert}(s, s_j, s_{j+1})$
18: **retour** Ind

5.3.5 Mutations

5.3.5.1 Mutation par ajout d'état

L'ajout (insertion) d'un état s dans une série d'états $Ind = \{s_1, ..., s_n\}$ entre deux états consécutifs s_i, s_{i+1} de Ind consiste à insérer l'état s construit à l'aide des atomes des états s_i et s_{i+1} de telle sorte que le problème $\Pi_i = \langle A, O, \widehat{s_i}, s_{i+1} \rangle$ soit intuitivement plus difficile (pour le planificateur embarqué) que les problèmes $\Pi'_i = \langle A, O, \widehat{s_i}, s \rangle$ et $\Pi'_{i+1} = \langle A, O, \widehat{s}, s_{i+1} \rangle$. L'algorithme 5 présente l'implantation de cette mutation. Pour la construction d'état s nous allons regrouper l'ensemble des atomes de s_i et s_{i+1} et les séparer en trois sous-ensembles : le sous-ensemble des atomes communs (A_c - ligne 3), le sous-ensemble des atomes *mutex* (A_m - ligne 2) et les autres (A_r - ligne 4). Tout se passe comme si le sous-ensemble A_m était ensuite séquentiellement partitionné en sous-ensembles regroupant chacun tous les mutex d'un atome donné de A_m. Le nombre d'atomes composant l'état s est le nombre d'atomes communs A_c (ligne 5) auquel on ajoute le nombre de groupes d'atomes mutex de A_m (lignes 6-9) plus un nombre aléatoire d'atomes pris dans A_r (lignes 10-16). s est alors construit par un choix uniforme d'un atome dans chaque groupe mutex et un choix uniforme d'atomes de A_r.

5.3.5.2 Mutation par ajout d'atome

La modification d'un état s d'une séquence d'états Ind consiste (cf. algorithme 6), soit à remplacer un atome donné a (ligne 3 - choisi uniformément) par un de

Algorithme 6 AjoutAtomesAveugle($Ind = (s_i)_{i \leq n}$)

Précondition : p_c, p_a // probabilités relatives pour changer ou ajouter un atome

1: **pour tout** $k \in [1, min(n, dernier(Ind) + 1)]$ **faire**
2: **si** $\mathbb{U}([0,1]) < \frac{p_c}{n}$ **alors** // modifier un atome
3: $a \leftarrow \mathbb{U}(s_k)$
4: $b \leftarrow \mathbb{U}(\{b \in voisin\ de\ a \wedge \nexists c \in s_k \setminus \{a\} \mid b \in M(c)\})$
5: $s_k \leftarrow (s_k \setminus \{a\}) \cup \{b\}$
6: **si** $\mathbb{U}([0,1]) < p_a$ **alors** // ajouter un atome
7: $a \leftarrow \mathbb{U}(\{b \in A \mid \nexists c \in s_k, b \in M(c)\})$
8: $s_k \leftarrow s_k \cup \{a\}$
9: **retour** Ind

ses voisins [2] (ligne 4) soit à ajouter un atome qui n'est en contradiction avec aucun atome composant l'état s. Un atome de A (respectivement de s) est ajouté (respectivement modifié) en fonction d'une probabilité relative p_a (respectivement p_c).

5.4 Construction orientée

Nous supposons ici que nous disposons d'une estimation du temps $T(a)$ d'apparition d'un atome a dans toute solution du problème initial Π. Ces estimations de dates peuvent être obtenues à l'aide d'une fonction heuristique admissible (par exemple la famille d'heuristiques h^m [Haslum 2000]) ou non admissible. Elles sont en général pré calculées par le planificateur embarqué. Introduite initialement dans [Bibai 2009b] et [Bibai 2009c], la méthode de construction orientée date des séries d'états a été formalisée et présentée à la communauté planning dans [Bibai 2010a].

5.4.1 Choix des atomes autorisés

Lors de nos expériences avec l'approche aveugle historique de construction des séquences d'états, nous avons observé sur plusieurs domaines de planification temporelle des benchmarks IPC, qu'il existait une corrélation entre le choix des atomes et la qualité finale des résultats obtenus [Bibai 2009c] (voir annexes). Ce qui montrait empiriquement que l'ensemble des prédicats autorisés pour la construction des séquences d'états est un paramètre très important de l'algorithme. Fort de ce constat, nous avons proposé une méthode basée sur les estimations des temps d'apparition des atomes dans toute solution du problème initiale Π afin de s'affranchir de ce choix critique. L'idée est de restreindre l'ensemble des atomes de base à ceux apparaissant à des dates estimées, et ensuite, de préserver la cohérence temporelle des séquences d'états tout au long de l'algorithme pour ainsi respecter la chronologie estimée entre les atomes. Un état d'une séquence donnée sera construit à une

[2] les atomes construits à partir de a grâce à la variation d'un des arguments du prédicat utilisé lors de la construction de a

date donnée en choisissant de manière uniforme un certain nombre d'atomes dans la restriction correspondante.

5.4.2 Dimensionnement

La taille maximale des séquences d'états à l'étape d'initialisation est égale au nombre de dates discrètes (noté $\#T$ avec $T = \{T(a) \neq 0 | a \in A\}$) fournies par l'estimateur des dates au plus tôt d'apparition des atomes. Après la phase d'initialisation, la taille des séquences d'états va éventuellement croître pour atteindre au plus une limite de $3 * \#T$ fixée empiriquement. En ce qui concerne la taille d'un état, étant donné qu'il est construit à une date donnée, sa taille maximale sera donc la taille maximale des états partiels qu'il est possible de construire suivant la restriction de A correspondante.

5.4.3 Initialisation

L'algorithme 7 présente l'implantation de l'initialisation orientée. Tout d'abord la taille N de la séquence d'états est choisie de façon uniforme entre $[1, \#T]$ (ligne 1). Ensuite, N dates sont choisies uniformément et ordonnées parmi les dates estimées possibles (lignes 4-7). Pour chaque date choisie, un sous-ensemble de l'ensemble des atomes de base est construit (A_t - ligne 11). L'état correspondant à cette date est alors construit à chaque sous-ensemble de manière progressive (lignes 13-17). Un atome est tout d'abord choisi de manière uniforme dans le sous-ensemble correspondant et ajouté à l'état. Ensuite, cet atome ainsi que tous les atomes mutuellement exclusifs avec lui, seront retirés du sous-ensemble. Le processus sera répété jusqu'à atteindre une borne initialement choisie dans l'intervalle $[1, card_{A_t}]$ (ligne 12), ou un sous-ensemble vide.

5.4.4 Croisement

Comme dans le paragraphe 5.3.4, l'opérateur de croisement devra essayer de conserver l'ordre entre les séquences d'états apprises au cours de l'évolution, mais aussi garantir la cohérence temporelle des séquences d'états résultantes. Nous avons dont opté pour un croisement à 1-point avec une seule séquence d'états résultante de l'opération. Il diffère du croisement à 1-point précédent (cf. 5.3.4) par le fait qu'il conserve la cohérence temporelle de la séquence d'états résultante. L'algorithme 8 présente son implantation. Tout d'abord, un état est uniformément choisi dans chacune des deux séquences (lignes 1 et 2). Enfin, la séquence d'état résultante de l'opération de croisement est construite de façon à respecter sa cohérence temporelle (lignes 3-6).

Algorithme 7 InitialisationOrientee

1: $N \leftarrow \mathbb{U}[1, \#T]$ // taille de la séquence d'états
2: $D \leftarrow \{\}$ // séquence ordonée de dates
3: **répéter**
4: $t \leftarrow \mathbb{U}(T)$
5: $T \leftarrow T \setminus \{t\}$
6: Insert(t, D) // maintenir l'orde dans D
7: **jusqu'à** $\#D = N$
8: Ind $\leftarrow \{\}$ // debut de construction d'une séquence
9: **pour** $t \in D$ **faire**
10: $s \leftarrow \{\}$ // debut de construction d'un état
11: $A_t \leftarrow \{a \in A \mid T(a) = t\}$ // atomes présents à t
12: $n \leftarrow \mathbb{U}([1, \#A_t])$ // taille de l'état
13: **tant que** $n \neq 0 \wedge A_t \neq \{\}$ **faire**
14: $a \leftarrow \mathbb{U}(A_t)$ // choix uniforme dans A_t
15: $s \leftarrow s \cup \{a\}$ // ajouter à s
16: $A_t \leftarrow A_t \setminus (\{a\} \cup M(a))$ // retirer tous les mutex
17: $n \leftarrow n - 1$
18: Ind $\leftarrow$ Ind $+ \{s\}$ // ajouter à la fin de la séquence
19: **retour** Ind

Algorithme 8 CroisementOrientée(Ind$_1$,Ind$_2$)

1: $s_{i_1} \leftarrow \mathbb{U}(Ind_1)$ // Ind$_1 = (s_i)_{1 \leq i \leq n}$
2: $t_{i_2} \leftarrow \mathbb{U}(Ind_2)$ // Ind$_2 = (t_i)_{1 \leq i \leq m}$
3: **si** $\Delta(t_b) > \Delta(s_a)$ **alors**
4: **retour** $(s_1, \ldots, s_a, t_b, \ldots, t_m)$
5: **sinon**
6: **retour** $(t_1, \ldots, t_b, s_a, \ldots, s_n)$

5.4.5 Mutations

5.4.5.1 Mutation par ajout d'état

Cet opérateur permet d'insérer dans une séquence d'états $Ind = \{s_1, ..., s_n\}$, un état s garantissant la cohérence temporelle de la séquence résultante. Lors de l'initialisation, nous avons supposé que toutes les dates estimées sont égales aux dates réelles et les séquences d'états ont été construites uniquement avec des atomes apparaissant à ces dates estimées fixes. Pour cet opérateur, nous allons supposer que ces dates estimées sont différentes des dates réelles. Nous supposerons donc que la date réelle correspondante à une date estimée donnée se trouve dans un intervalle de temps que nous appellerons *Voisinage temporel* (cf. section 5.1.2.3) de la date estimée considérée. Un état s sera ensuite construit dans l'intervalle de temps correspondant, à une date estimée choisie dans la liste, et ajouté à Ind. L'algorithme 9 présente l'implantation de cet opérateur. Tout d'abord deux états consécutifs sont choisis uniformément (ligne 1). Ensuite, une date t est choisie de manière à respecter la cohérence temporelle (ligne 3). Une restriction de l'ensemble des atomes est alors construite à partir de t en tenant compte du rayon d'incertitude (ou de voisinage)

Algorithme 9 AjoutEtatOrientée($Ind = (s_i)_{i \leq n}$)

Précondition : r // rayon de voisinage

1: $j \leftarrow \mathbb{U}([1,$ min(n,dernier(Ind))$])$
2: $s \leftarrow \{\}$ // inserer s entre s_j et s_{j+1}
3: $t \leftarrow \mathbb{U}(\{t \in T \mid \Delta(s_j) < t \leq \Delta(s_{j+1})\})$
4: $A_t \leftarrow \{a \in A \mid T(a) \in N_t(t, r)\}$
5: $A_m \leftarrow \{\}$ // ensemble d'atomes non 2 à 2 mutuellement exclusifs
6: **tant que** $A_t \neq \{\}$ **faire**
7: $\quad a \leftarrow \mathbb{U}(A_t)$
8: $\quad A_m \leftarrow A_m \cup \{a\}$
9: $\quad A_t \leftarrow A_t \setminus (\{a\} \cup M(a))$
10: $N \leftarrow \mathbb{U}([1, \#A_m])$ // taille de l'état
11: **répéter**
12: $\quad a \leftarrow \mathbb{U}(A_m)$ // choix uniforme d'un atome de A_m
13: $\quad s \leftarrow s \cup \{a\}$ // l'ajouter à s
14: $\quad A_m \leftarrow A_m \setminus \{a\}$ // le retirer de A_m
15: **jusqu'à** $card_s = N$
16: insert(Ind, s, j) // inserer s avant j
17: **retour** Ind

r donné par l'utilisateur (A_t - ligne 4). A_t est ensuite partitionné en sous-ensembles d'atomes mutuellement exclusifs A_m. La taille de l'état intermédiaire s, sera choisie entre 1 et $card_{A_m}$. Ensuite, $card_s$ sous-ensembles de A_m seront choisis uniformément et un atome par sous-ensemble sera choisi uniformément et ajouté à s. Enfin, s sera inséré dans Ind.

5.4.5.2 Mutation par ajout d'atome

Pour cet opérateur, nous allons supposer avoir commis des erreurs lors des différents choix d'atomes effectués au cours des phases d'initialisation et de mutation par ajout d'état. Cette mutation aura donc pour but de corriger (ou réparer) ces éventuelles erreurs sur les choix d'atomes. L'algorithme 10 présente l'implantation de cet opérateur. Pour chaque état $Ind[k]$ de la séquence Ind considérée, tout d'abord un atome a uniformément choisi sera éventuellement remplacé par un atome b de $M(a)$ (lignes 2-5), et enfin on ajoutera éventuellement à $Ind[k]$ un atome appartenant à un sous ensemble mutex non pris en compte lors des phases d'initialisation et de mutation (lignes 6-8).

5.5 Conclusion

Dans ce chapitre, nous avons vu en détail les différentes approches de construction des séquences d'états (aveugle et orientée) ainsi que les pseudo-codes des algorithmes permettant leurs implantations. A sa lecture, on constate l'importance du grand nombre de paramètres à fixer par l'utilisateur afin de garantir le bon fonctionnement de l'algorithme. De quelle façon pourrait-on choisir automatiquement

Algorithme 10 AjoutAtomeOrientée($Ind = (s_i)_{i \leq n}$)

Précondition : p_c, p_a // probabilités d'ajout et de modification

pour tout $k \in [1,\min(n, dernier(Ind))]$ **faire**
  si $\mathbb{U}([0,1]) < \frac{p_c}{n}$ **alors** // modifier un atome ?
    $a \leftarrow \mathbb{U}(\text{Ind}[k])$
    $b \leftarrow \mathbb{U}(\{b \in M(a) \mid T(b) = \Delta(\text{Ind}[k]) \wedge \nexists c \in (\text{Ind}[k] \setminus \{a\}), b \in M(c)\})$
    $\text{Ind}[k] \leftarrow (\text{Ind}[k] \setminus \{a\}) \cup \{b\}$
  si $\mathbb{U}([0,1]) < p_a$ **alors** // ajouter un atome ?
    $a \leftarrow \mathbb{U}(\{b \in A \mid T(b) = \Delta(\text{Ind}[k]) \wedge \nexists c \in \text{Ind}[k], b \in M(c)\})$
    $\text{Ind}[k] \leftarrow \text{Ind}[k] \cup \{a\}$
retour Ind

ces paramètres ? Cette question sera traitée dans le prochain chapitre.

Chapitre 6

Paramètres et Réglage Automatique

Sommaire

Le réglage des paramètres a toujours été considéré comme le talon d'Achille des algorithmes évolutionnaires à cause du manque de fondements théoriques y référant. Les méthodes employées révèlent donc des essai-erreurs, ce qui entraine un énorme coût de calcul. Certes, l'efficacité des algorithmes évolutionnaires a été démontrée sur de très nombreux problèmes réels [Yu 2008]. Mais ces résultats ont le plus souvent été obtenus avec des ensembles de valeurs de paramètres (ou configurations de paramètres) résultant de plusieurs tests et comparaisons. Afin de réduire la durée globale de l'opération, plusieurs méthodes ont été proposées et leur abondance a permis l'émergence d'un nouveau et important domaine de recherche [Lobo 2007].

D'après [Eiben 2007], on distingue deux types d'approches de réglage automatique des algorithmes évolutionnaires : le "*Parameter tuning*" qui se fait avant l'exécution de l'algorithme et le "*Parameter control*" qui modifie et adapte les paramètres au cours de l'exécution de l'algorithme en fonction du comportement de l'algorithme.

Dans le but de comparer l'approche $D\&E$ à l'état de l'art des méthodes de résolution directes des problèmes de planification, et parce que le "*Parameter control*" est un domaine de recherche à part entière, nous avons opté pour les approches dites de "*Parameter tuning*". Une méthode naïve pour régler les paramètres d'un algorithme évolutionnaire donné sur une classe de problèmes est d'utiliser par exemple un plan d'expérience (eventuellement factoriel) pour tous ces problèmes, avec plusieurs exécutions pour chaque configuration de paramètres du fait du caractère stochastique de l'algorithme, et d'en extraire les meilleures configurations significatives à l'aide d'une méthode statistique classique par exemple l'ANOVA (*Analysis of Variance*). Les méthodes de "*Parameter tuning*" vont quant à elles essayer de réduire la durée globale d'exécutions des méthodes naïves en focalisant les efforts[1] sur les configurations de paramètres les plus prometteuses. Depuis les travaux de [Grefenstette 1986] plusieurs méthodes ont été proposées parmi les plus récentes *REVAC* [Nannen 2007], *SPO* [Bartz-Beielstein 2005, Hutter 2009], *Racing* [Birattari 2002] et *Meta-EAs+racing* [Yuan 2007].

6.1 Quelques méthodes de "*Parameter tuning*"

6.1.1 REVAC

La méthode *REVAC* (ou "*Relevance Estimation and Value Calibration*") utilise la théorie de l'information pour mesurer la sensibilité d'un paramètre par rapport au choix de ses valeurs. Au lieu d'estimer la performance d'un algorithme pour plusieurs valeurs d'un paramètre, REVAC estime les performances attendues lorsque ces valeurs sont choisies à partir d'une distribution de probabilité dont la densité maximise l'entropie de Shannon[2]. Dans la pratique *REVAC* est un algorithme évolutionnaire qui utilise la maximisation de l'entropie de Shannon dans son opérateur de mutation afin de générer un nouvel individu qui maximise les performances attendues de l'algorithme.

REVAC commence par générer de manière stochastique un ensemble de solutions potentielles au problème posé, ici un ensemble de configurations de paramètres. Cette population initiale est ensuite évaluée à l'aide de l'algorithme évolutionnaire considéré. Un sous ensemble de configurations de paramètres est sélectionné par la suite de manière déterministe. Afin de créer un nouvel individu, le sous ensemble de configurations de paramètres sélectionné sera ordonné de manière croissante des valeurs d'un paramètre choisi pour être muté. Une densité de probabilité uniforme sera ensuite définie[3] pour chaque paire de valeurs consécutives de ce paramètre. En-

[1] nombre d'exécution d'un ensemble de valeurs de paramètres

[2] Soit D une densité de probabilité. L'entropie de Shannon s'écrit

$$H(D) = -\int Dlog(D)$$

[3] Soit x et y deux valeurs consécutives et n le nombre d'intervalles entre x et y. Alors la densité $D = \frac{n}{|x-y|}$

fin, une nouvelle valeur de ce paramètre maximisant l'entropie de Shannon de cette densité de probabilité sera potentiellement choisie entre ces deux valeurs consécutives.

6.1.2 SPO

La méthode *SPO* ou *Sequential Parameter Optimisation* utilise le *Design and Analysis of Computer Experiments (DACE)* (ou modèle spatial adapté du modèle de Krigeage habituellement utilisé en géostatistique) afin de prédire la meilleure configuration de paramètres d'un algorithme. A partir d'un ensemble de configurations de paramètres uniformément distribuées sur chaque axe du domaine de définition des paramètres et générées à l'aide d'un plan à marges uniformes (les hypercubes latin[4]), *SPO* sélectionne itérativement les hypercubes optimisant la performance prédite de l'algorithme. Après avoir généré un ensemble de configurations de paramètres initial, l'algorithme est exécuté une fois avec ces différentes configurations. Ensuite, le *DACE* est utilisé afin de déterminer (ou prédire) les configurations de paramètres optimisant les performances attendus de l'algorithme, ceci en fonction des résultats obtenus. L'ensemble des configurations de paramètres initial est ensuite mis à jour avec une partie des meilleures configurations de paramètres ainsi produite. Enfin, le processus, exécution de l'algorithme sur le nouvel ensemble de configurations de paramètres, prédiction des meilleures configurations, et mis-à-jour de l'ensemble initial, est répété jusqu'au critère d'arrêt. Le critère d'arrêt de *SPO* est, soit un nombre maximal d'exécution de l'algorithme pour chaque meilleure configuration de paramètres, soit l'obtention de la configuration optimale. Il est à noter qu'après l'étape d'exécution, l'algorithme a été exécuté un même nombre de fois sur toutes les configurations de l'ensemble initial mise-à-jour.

6.1.3 Racing

Le *Racing* est une méthode statistique proposée initialement pour la sélection de modèles en apprentissage artificiel [Maron 1994]. Elle a été introduite dans le domaine des algorithmes évolutionnaires par [Birattari 2002] dans le but de focaliser rapidement la recherche des meilleures configurations de paramètres sur les plus performants. A partir d'un plan d'expérience donné, l'idée est d'identifier grâce à un test statistique les mauvaises configurations de paramètres après un petit nombre d'exécution de l'algorithme pour chaque configuration de paramètres, et de ne continuer progressivement l'exécution de l'algorithme qu'avec les configurations de paramètres les plus performants : après chaque exécution, toutes les configurations de paramètres sont comparées à la meilleure configuration, et les configurations

[4]Soit n le nombre de valeurs d'un domaine et soit d le nombre de paramètres d'un algorithme. L'hypercube latin est une matrice de n lignes et d colonnes. Chaque axe du domaine est divisé en n segments de même longueur d de façon à obtenir un maillage du domaine. L'hypercube latin est alors obtenu en sélectionnant n points parmi les n^d points de la grille de façon à ce que chaque colonne soit une permutation de n symboles quelconques.

de paramètres significativement mauvaises sont supprimées. Le cycle d'exécution-comparaison-suppression est ensuite répété jusqu'à ce que soit une seule configuration de paramètres, soit un nombre maximal fixé a priori d'exécutions de l'algorithme considéré ait été exécuté pour chaque meilleure configuration de paramètres.

6.1.4 Meta-EAs + Racing

Est une méthode hybride combinant les algorithmes évolutionnaires et la méthode *Racing*. Un individu ou une solution candidate sera ici une configuration de paramètres. Un individu est généré à chaque étape de l'algorithme évolutionnaire à partir d'un parent à l'aide d'une distribution gaussienne centrée sur la configuration de paramètres du parent sélectionné pour être modifié. Pour la sélection des parents, le *Racing* sera appliqué à l'algorithme à régler afin d'en extraire les meilleures configurations de paramètres (ou individus) de la génération suivante. La population initiale est générée aléatoirement.

Les différentes méthodes de réglage automatique des paramètres ont été comparées et testées sur les fonctions de Rastrigin [Smit 2009]. Leurs résultats montrèrent que de celles citées ci-dessus, Méta-EAs + Racing était la meilleure méthode de réglage des paramètres. Cependant, le coût de l'évaluation des configurations de paramètres de Meta-EAs + Racing est très importante [Yuan 2007]. De plus, elle est inefficace sur des configurations de paramètres discrets. Par conséquent, de toutes ces méthodes, le Racing semble être la plus robuste car elle ne fait aucune hypothèse sur la forme (ou le modèle) de l'espace de recherche des configurations de paramètres ni sur le type (continus ou discrets) de paramètres. Par contre, elle a l'inconvénient de ne pouvoir donner que la meilleure configuration de paramètres parmi un ensemble fixé a priori.

6.2 Les paramètres des systèmes Divide-And-Evolve

Les paramètres des systèmes Divide-And-Evolve peuvent être partitionnés en trois sous ensembles :

- ceux permettant le paramétrage du planificateur embarqué,
- ceux communs aux deux approches de construction des séries d'états (cf. chapitre 5),
- et ceux spécifiques à chaque approche de construction des séries d'états.

6.2.1 Les paramètres spécifiques au planificateur embarqué

Les paramètres spécifiques au planificateur embarqué sont utilisés lors de la phase de *préprocessing*[5] et lors du calcul de la fitness. Ce sont les paramètres utilisés par

[5]La phase dite de *preprocessing* permet de construire des structures de données utilisées par les systèmes Divide-and-Evolve et le planificateur embarqué choisit à partir des données du problèmes.

le planificateur embarqué choisi pour la résolution d'un problème de planification donné auxquels on ajoutera un paramètre (ou *mesure de difficulté*) permettant d'éliminer les problèmes plus difficiles que le problème initial que l'on désire résoudre. Ce sera par exemple pour le planificateur YAHSP, le **nombre de noeuds** (cf. section 2.3.1), et pour le planificateur CPT le **nombre de backtracks** (cf. section 2.4.6). Pour les autres paramètres, nous utiliserons ceux préconisés par les différents auteurs lors des compétitions IPC.

6.2.2 Les paramètres communs

Les paramètres communs aux deux approches de construction des séquences d'états sont :

- les paramètres génériques de l'algorithme évolutionnaire[6],
- les probabilités de croisement (P_{cross}) et de mutation (P_{mut}),
- les probabilités de modification (p_c) et d'ajout (p_a) d'un atome utilisées dans les mutations par ajout d'atome(cf. chapitre 5 algorithmes 6 et 10)
- et les poids relatifs des différents opérateurs de mutation

Les poids relatifs associées à chaque opérateur de mutation sont :

- $w_{addState}$ pour la mutation par ajout d'un état,
- $w_{delState}$ pour la mutation par suppression d'un état,
- $w_{addAtom}$ la mutation par ajout d'un atome,
- et $w_{delAtom}$ la mutation par suppression d'un atome.

6.2.3 Les paramètres spécifiques aux approches de construction des séries d'états

Les paramètres spécifiques à l'approche de construction aveugle des séries d'états sont les tailles maximales d'un état et celle d'une série d'états.

En ce qui concerne l'approche de construction orientée des séries d'états les paramètres spécifiques utilisées sont le rayon de voisinage r de la mutation par ajout d'état (cf. chapitre 5 algorithme 9), et l'heuristique utilisé pour le calcul des dates au plus tôt des atomes.

6.2.4 Les paramètres à régler

Compte tenu du nombre important des paramètres à régler, nous avons décidé tout d'abord de fixer une fois pour toutes les paramètres génériques de l'algorithme évolutionnaire. Comme il est impossible de garantir qu'un enfant serait toujours meilleur que son (ou ses) géniteur(s), nous avons opté pour une stratégie d'évolution de type $(\mu + \lambda) - ES$.

La figure 6.1 montre sur le problème `zeno 11` les distributions des solutions de Divide-and-Evolve obtenues en variant les valeurs de μ et λ. On remarque que la

[6] le moteur d'évolution, le type de sélection et de remplacement, et le critère d'arrêt

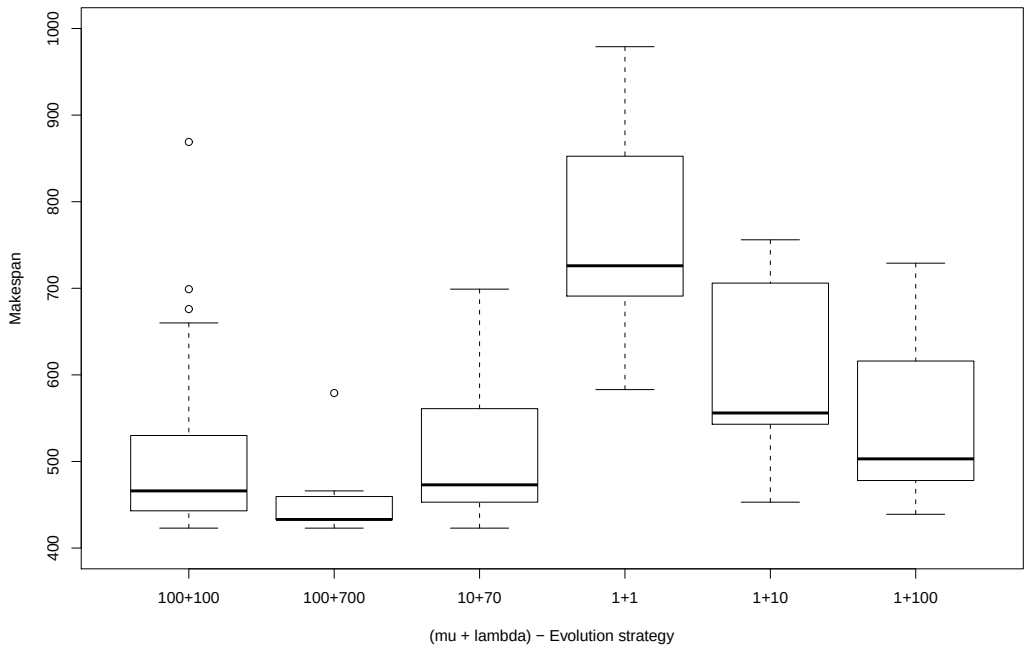

FIG. 6.1 – Variation des stratégies d'évolutions : distribution des qualités de 11 solutions de DAEX+YAHSP (cf. chapitre 7 section 7.6) avec la configuration de paramètres du chapitre 7 sur l'instance 11 du problème `zeno`. Valeur calculée par YAHSP : 1715.

stratégie d'évolution (100+700)-ES a la plus petite variance parmi les autres stratégies testées. De plus, la qualité des solutions trouvée avec la stratégie d'évolution (100+700)-ES est en moyenne meilleure que ceux des autres stratégies testées. Par conséquent, nous avons retenu pour les paramètres génériques de l'algorithme évolutionnaire :

- moteur d'évolution : $(100 + 700) - ES$
- critère d'arrêt : 50 générations sans amélioration de la fonction objective avec une borne maximale fixée à 1000 générations
- sélection : tournoi déterministe à 5 individus
- remplacement : weakElitism - le meilleur parent remplace un enfant si nécessaire.

Concernant les paramètres spécifiques au planificateur embarqué, Caractériser la difficulté (ou la facilité) de résolution d'un problème de planification donné n'est pas trivial. Comme il n'existe pas à notre connaissance une méthode permettant de caractériser les problèmes de planification, nous avons opté pour un sondage de la population initiale. La *mesure de difficulté* autorisée pour la résolution d'une instance donnée sera fixé en deux étapes. Dans un premier temps, la population initiale

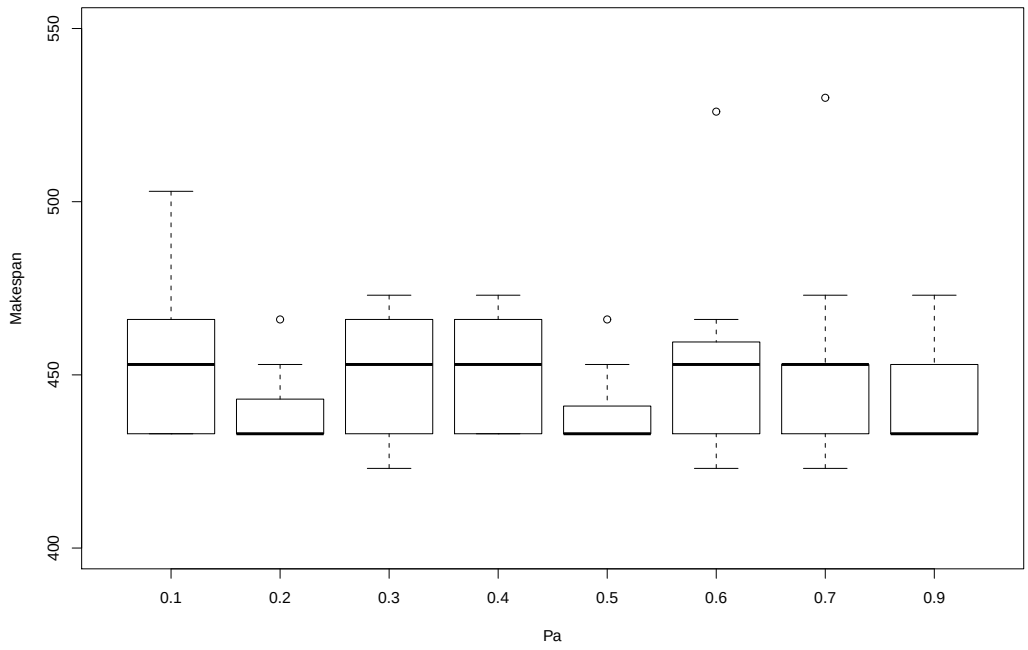

FIG. 6.2 – Variation de P_a : distribution des qualités de 11 solutions de DAEX+YAHSP (cf. chapitre 7 section 7.6) avec la configuration de paramètres du chapitre 7 sur l'instance 11 du problème `zeno`. Valeur calculée par YAHSP : 1715.

sera évaluée avec une très faible contrainte sur la *mesure de difficulté*[7]. Ensuite, pour le reste de l'algorithme, la *mesure de difficulté* sera fixée à la médiane des valeurs effectivement utilisées pour la résolution des individus faisables de la population initiale. Pour les autres paramètres, nous avons choisi d'utiliser les paramètres par défaut du planificateur embarqué choisi.

Enfin, la probabilité P_a des opérateurs de mutations par ajout d'atomes des deux différentes approches de construction des séries d'états a été fixée à 0.5. En effet, comme le montre la figure 6.2 sur le problème `zeno 11`, P_a semble avoir peu d'influence sur les résultats, et $P_a = 0.5$ est un bon compromis pour Divide-and-Evolve.

L'opération de réglage des paramètres se fera sur les paramètres restant à savoir les 3 probabilités $\{P_{cross}, P_{mut}, p_c\}$, les 4 coefficients de proportionnalité associés à chaque opérateur de mutation $\{w_{addState}, w_{delState}, w_{addAtom}, w_{delAtom}\}$, le rayon de voisinage r de la mutation par ajout d'état de l'approche de construction orientée des série d'états[8], et les tailles maximales des états et des séries d'états pour l'approche

[7] Ce sera par exemple pour le planificateur YAHSP un grand nombre de noeuds

[8] l'heuristique utilisée pour le calcul des dates est fixée une fois pour toute avant la recherche de

de construction aveugle des séries d'états.

6.3 Le Racing appliqué à Divide-And-Evolve

Après avoir constaté que la durée d'exécution de l'algorithme Divide-and-Evolve pouvait, en fonction des configurations de paramètres choisies, passer de quelques minutes à plusieurs jours de calcul, nous avons opté pour la méthode *racing*. En effet, le *racing* permet par la définition du plan d'expérience, de borner la durée maximale de l'opération de réglage des paramètres.

L'efficacité du *racing* est dépendante du type de test statistique utilisé pour la comparaison des configurations de paramètres. Comme aucune hypothèse ne peut être faite sur la forme des distributions sous-jacentes des résultats, nous avons choisi d'utiliser le test d'analyse de variance de Friedman comme évoqué dans les travaux de [Birattari 2002, Yuan 2004, Yuan 2007]. Comme tout test non paramétrique, le test d'analyse de variance de Friedman travaille non pas sur les valeurs numériques des observations, mais sur leurs rangs. Une fois ces observations convenablement réunies dans un tableau approprié, chaque observation est classée sur une échelle numérique de k valeurs (rang de 1 à k). Le test s'appuie alors sur ces rangs pour accepter ou rejeter l'hypothèse nulle selon laquelle k échantillons appariés proviennent de la même configuration de paramètres.

Cependant, quel que soit le type de test statistique choisi, nous devons aussi choisir le nombre d'exécutions initial significatif de chaque configuration de paramètres ainsi que le *niveau de signification (ou niveau de risque)* afin de ne pas éliminer précocement une bonne configuration de paramètres.

Après avoir effectué plusieurs tests, nous avons constaté qu'il serait préférable de commencer l'élimination initiale des configurations de paramètres à partir de **11** exécutions de l'algorithme. En effet, si on regarde par exemple la distribution des qualités de 50 solutions de Divide-and-Evolve obtenu sur l'instance 30 du problème `gold-miner`[9] (cf. figure 6.3), on peut remarquer que les limites inférieures adjacentes[10] de toutes les configurations de paramètres coïncident. Il faudrait donc choisir un nombre d'exécutions initial significatif afin de ne pas éliminer précocement une bonne configuration de paramètres.

Le *niveau de risque* a été quant à lui fixé à 0.025 afin d'avoir une forte contrainte sur le rejet d'une mauvaise configuration de paramètres.

Une observation sera mesurée en fonction de la qualité de la solution obtenue par le système Divide-and-Evolve pour une configuration de paramètres donnée et la durée d'exécution de l'algorithme. Cette mesure est un rang bi-critère. Le premier critère représente la qualité de la solution obtenue et le second le temps d'exécution de l'algorithme. Enfin, nous avons décidé d'arrêter le *racing* après un total de **50** exécutions de l'algorithme pour chaque meilleure configuration de paramètres.

la meilleure configuration de paramètres.

[9] http ://eecs.oregonstate.edu/ipc-learn/pages/domain_descriptions.htm

[10] Différence entre le premier quartille et 1.5 fois l'écart inter quartille

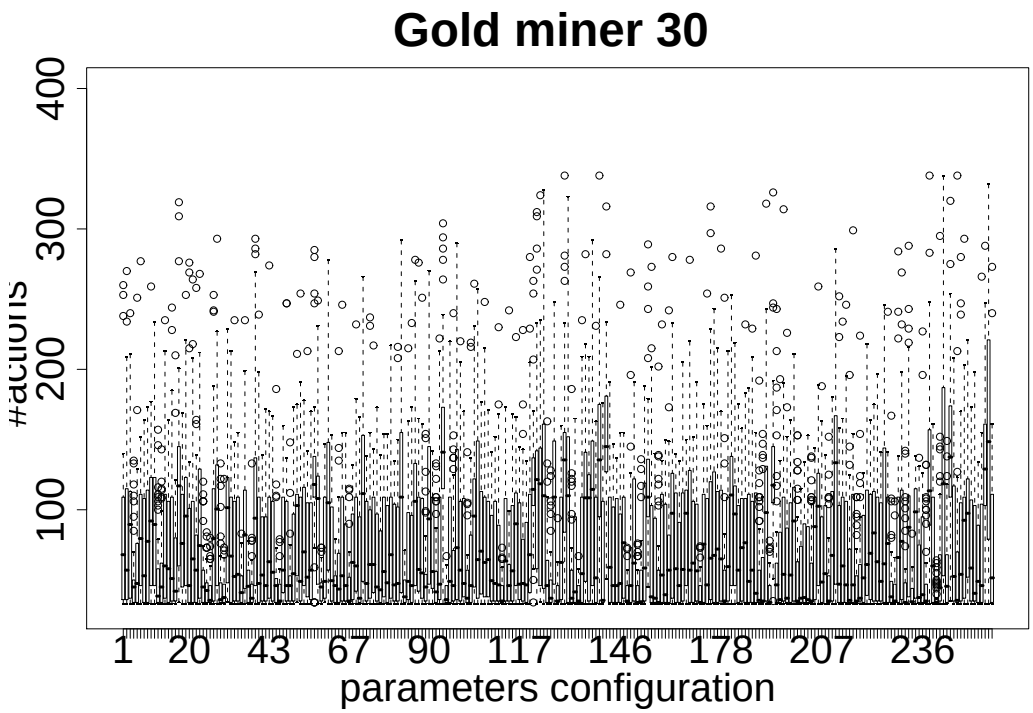

FIG. 6.3 – Distribution des qualités de 50 solutions de 256 configurations de paramètres sur instance 30 du problème `gold-miner`.

6.3.1 Choix du plan d'expériences

Le choix du plan d'expérience est une partie délicate de cette méthode. Dans le but de réduire de durée globale de l'opération de réglage des paramètres, nous avons proposé de restreindre l'ensemble des configurations de paramètres initiaux aux meilleures configurations de paramètres obtenu sur certains domaines [Bibai 2009a]. Pour expliquer cette restriction, nous avons conjecturé une similitude du comportement des systèmes Divide-and-Evolve sur des domaines différents. En d'autres termes, une bonne configuration de paramètres sur un domaine donné pourrait aussi l'être sur un autre domaine. Néanmoins, nous avons aussi testé l'utilisation d'un plan d'expérience factoriel [Bibai 2010c] afin de vérifier que les configurations de paramètres obtenus avec notre restriction donnaient des résultats comparables avec ceux du plan factoriel.

Cependant, quelque soit le plan d'expérience choisit, nous devons aussi choisir un ensemble d'instances représentatives du domaine (ou des domaines) sur lesquelles sera réglé l'algorithme, car le test d'une configuration de paramètres pour la résolution d'une instance donné peut éventuellement prendre beaucoup de temps.

6.3.1.1 Plan d'expériences

Pour la définition du plan d'expériences, 2 valeurs par paramètre ont été choisies, soit au total 256 configurations de paramètres (un plan d'expérience plus fourni aurait mémé à une très longue durée des expériences). Les valeurs suivantes ont été utilisées : $P_{cross} = 0.2$ ou 0.6, $P_{mut} = 0.6$ ou 0.8, les poids relatifs des 4 opérateurs de mutation $w_{(*)} = 1$ ou 3, le rayon de voisinage $r = 1$ ou 2, et la probabilité de modifier un atome dans un état subissant une mutation $p_c = 0.2$ ou 0.8.

6.3.1.2 Généralité des paramétrages

Serait-il donc raisonnable de supposer une similitude entre tous les domaines quel que soit le type de problème de planification (ou *racing* global) ? Ou encore une similitude entre les instances (ou les problèmes) du même domaine (ou *racing* par domaine) ? ou pas de régularité du tout (ou *racing* par instance). Ces questions seront traitées dans la section suivante.

Afin de choisir les instances représentatives d'un domaine, nous avons opté pour la stratégie suivante : tout d'abord, nous allons essayer de résoudre toutes les instances d'un domaine avec le planificateur embarqué, ceci avec un temps limité (par exemple 10 minutes). Enfin, nous allons choisir deux instances pour le *racing*, dont l'une sera choisie parmi les instances résolues par le planificateur embarqué (à moins que toutes les instances ne soient non résolues). L'instance résolue retenue est celle ayant le temps d'exécution le plus long et celle non résolue sera celle qui aura utilisé le plus petit espace mémoire lors de la phase de prétraitement. Pour les instances représentatives de plusieurs domaines, une instance par domaine sera retenue parmi les deux instances représentatives dudit domaine. L'instance choisie par domaine sera l'instance la plus difficile[11].

6.4 Résultats et discussions

Dans ce qui suit nous présenterons les conséquences du choix de ces hypothèses sur le réglage des paramètres de l'approche orientée date de construction des séries d'états uniquement.

Pour les résultats de ce chapitre, le planificateur approché YAHSP[Vidal 2004b] a été embarqué pour la résolution des séries d'états. La comparaison a été faite avec l'état de l'art des meilleurs systèmes de la littérature pour chaque type de problème de planification. Les tests ont quant à eux été effectués sur des machines Linux de 2GHz de fréquence d'horloge, 6Mb de mémoire cache et 16 Gb de mémoire vive (ou RAM) sur les domaines `gold-miner` (planification STRIPS classique), `zeno simple time` (planification temporelle) et `openstacks-ipc6-satisficing-track` (planification avec coûts) des benchmarks de la compétition internationale de planification

[11] L'instance la plus difficile sera celle non résolue par le planificateur embarqué choisi ou celle ayant utilisé le plus grand espace mémoire dans le cas où les deux instances représentatives du domaine n'auraient pas été résolues.

(IPC).

6.4.1 Résultats

Les tables 6.1, 6.3 et 6.2 montrent les résultats de Divide-and-Evolve pour chacune des hypothèses et ceux de l'état de l'art des planificateurs (cf. chapitre 2 section 2.5) : le planificateur optimal CPT [Vidal 2006] pour le domaine `gold-miner`, le planificateur approché LPG [Gerevini 2003a, Gerevini 2003b] pour le domaine `zeno simple time`, et le planificateur approché LAMA [Richter 2008] pour le domaine `openstacks-ipc6-satisficing-track`. Pour les planificateurs stochastiques LPG et Divide-and-Evolve, ces tables montrent les meilleurs résultats de 11 exécutions de l'algorithme sur chaque instance des domaines considérés pour une durée d'exécution limitée à 30 minutes. Pour Divide-and-Evolve les configurations de paramètres utilisées sont celles obtenues après le *racing*.

Les figures 6.4, 6.6 et 6.5 montrent, pour chacune des hypothèses de régularité, le comportement de solutions de Divide-and-Evolve par rapport à celles de l'état de l'art. La figure 6.7 montre les différentes configurations de paramètres trouvées pour chacune des instances des domaines testés.

6.4.2 Discussions

La première observation significative qui découle de ces expériences est qu'il n'existe pas de régularité ni entre les instances d'un même domaine ni entres les instances de plusieurs domaines (voir par exemple les variations des configurations de paramètres trouvées après le *racing* en fonction des instances de la figure 6.7).

La seconde observation est que la variance ainsi que la qualité des solutions trouvées par Divide-and-Evolve augmente ou diminue en fonction des hypothèses de régularités choisies. La variance (respectivement la qualité) des solutions trouvées par Divide-and-Evolve sur une instance sera la plus importante (respectivement la plus mauvaise) pour le *racing* global(ou sur plusieurs domaines) et moins importante (respectivement la meilleure) pour le *racing* par instance (voir par exemple la figure 6.4). Le *racing* par instance trouve toujours des solutions de meilleure qualité que les autres types de *racing*. Néanmoins, pour les petites instances, par exemple les instances 1 à 9 du domaine `openstacks-ipc6-satisficing-track` (voir table 6.2 et figure 6.5), presque toutes les configurations de paramètres essayées permettent à Divide-and-Evolve d'obtenir la meilleure solution.

6.5 Conclusion

Après avoir passé en revue quelques méthodes permettant le réglage automatique *offline* des paramètres des algorithmes Divide-and-Evolve, nous avons listé les différents paramètres des systèmes Divide-and-Evolve et justifié le choix de la méthode *racing* qui nécessite la définition d'un plan d'expérience. Cependant, quel

que soit le plan d'expérience choisi, nous devons choisir un ensemble de problèmes représentatif du domaine sur lequel sera réglé l'algorithme.

Les résultats ont montré que la qualité des solutions obtenues par Divide-and-

#	Global racing		Domain racing		Instance racing		Best
Gold	Min	med.	Min	med.	Min	med.	CPT
1	**28**	31	**28**	31	**28**	31	28
2	**19**	21	**19**	21	**19**	21	19
3	**18**	21	**18**	20	**18**	20	18
4	**24**	28	**24**	27	**24**	25	[23]
5	**20**	20	**20**	20	**20**	20	20
6	**24**	26	**24**	26	**24**	26	24
7	**24**	29	**24**	28	**24**	28	24
8	**25**	27	**25**	27	**25**	27	25
9	**26**	30	27	29	**26**	29	26
10	**17**	19	**17**	19	**17**	19	17
11	**29**	33	**29**	33	**29**	33	29
12	**25**	32	**25**	32	**25**	33	25
13	**27**	34	**27**	32	**27**	29	27
14	**27**	28	**27**	28	**27**	27	27
15	**26**	34	**26**	32	**26**	32	26
16	**24**	27	**24**	27	**24**	27	24
17	**34**	41	**34**	39	**34**	39	[32]
18	**22**	32	**22**	31	**22**	30	22
19	**35**	38	36	38	**35**	38	[31]
20	**30**	38	**30**	35	**30**	38	30
21	**33**	35	**33**	46	**33**	34	31
22	30	51	29	43	**28**	41	28
23	33	43	**32**	46	33	43	32
24	**39**	55	**39**	52	**39**	52	39
25	**41**	173	**41**	62	**41**	50	[37]
26	36	50	36	50	**35**	50	[31]
27	39	50	39	49	**38**	46	[34]
28	**25**	40	**25**	41	**25**	36	25
29	30	41	**29**	38	**29**	38	29
30	**33**	130	**33**	34	**33**	34	[31]

TAB. 6.1 – Meilleures valeurs obtenues par Divide-and-Evolve après le *racing* et valeurs optimales de CPT pour chaque instance du domaine `gold-miner` pour chaque hypothèse de régularité. En gras les meilleurs valeurs trouvées par Divide-and-Evolve. Dans la dernière colonne, les valeurs optimales (trouvées par CPT). Les valeurs entre [] indiquent que Divide-and-Evolve n'a pas trouvé une solution optimale (7 instances sur 30, pas forcément les plus grandes - e.g. l'intance 4).

#	Global racing		Domain racing		Instance racing		Best
Open	Min	med.	Min	med.	Min	med.	LAMA
1	**2**	2	**2**	2	**2**	2	2
2	**3**	3	**3**	3	**3**	3	3
3	**2**	2	**2**	2	**2**	2	2
4	**2**	3	**2**	3	**2**	3	2
5	**2**	2	**2**	2	**2**	2	2
6	**4**	4	**4**	4	**4**	4	4
7	**4**	4	**4**	4	**4**	4	4
8	**4**	4	**4**	4	**4**	4	4
9	**4**	4	**4**	4	**4**	4	4
10	**4**	5	**4**	5	**4**	4	4
11	**5**	6	**5**	5	**5**	5	10
12	**3**	5	**3**	5	**3**	4	3
13	**5**	6	**5**	6	**5**	6	10
14	5	7	5	7	**4**	7	10
15	**5**	7	**5**	6	**5**	6	10
16	8	10	**7**	10	**7**	10	10
17	8	8	**7**	9	**7**	8	10
18	**6**	8	7	8	**6**	7	18
19	12	14	11	14	**10**	13	11
20	14	17	15	22	**12**	16	12
21	10	16	10	16	**9**	12	10
22	18	29	17	28	**16**	19	[14]
23	17	27	14	26	**13**	14	[10]
24	13	24	14	25	**12**	16	[11]
25	25	39	27	38	**21**	38	[20]
26	22	36	22	36	**18**	35	[15]
27	27	39	25	37	**22**	25	[21]
28	36	53	37	52	**33**	52	[32]
29	42	53	35	54	**33**	54	[30]
30	37	54	**33**	53	**33**	53	34

TAB. 6.2 – Meilleures valeurs obtenues par Divide-and-Evolve après le *racing* et LAMA pour chaque instance du domaine `openstacks-ipc6-satisficing-track` pour chaque hypothèse de régularité. En gras les meilleurs valeurs trouvées par Divide-and-Evolve dans la dernière colonne les valeurs trouvées par LAMA. Surlignées les valeurs de référence inférieures à celles trouvées par Divide-and-Evolve et les valeurs entre [] indique que Divide-and-Evolve n'a pas égalé la solution de référence. Pour la plupart des instances moyenne, Divide-and-Evolve est meilleur que LAMA, et pour la plupart des grandes instances, LAMA est meilleur que Divide-and-Evolve.

Evolve peut varier en fonction des hypothèses de régularité du domaine (ou des domaines). Néanmoins, les meilleures solutions obtenues par Divide-and-Evolve avec une configuration de paramètres issue du *racing* global (avec une hypothèse de régularité sur tous les domaines), sont le plus souvent proches, d'au moins à 90% des meilleures solutions de l'état de l'art. On peut donc sereinement, dans le but d'une comparaison avec l'état de l'art, envisager l'utilisation d'une seule configuration de paramètres pour tous les domaines bien que les résultats obtenus ne soient pas les meilleurs possibles. C'est ce qui sera fait plus en détails dans le chapitre suivant.

#	Global racing		Domain racing		Instance racing		Best
zeno	Min	med.	Min	med.	Min	med.	LPG
1	**173**	173	**173**	173	**173**	173	173
2	**592**	599	**592**	599	**592**	592	592
3	**280**	280	**280**	280	**280**	280	280
4	529	529	**522**	529	**522**	529	522
5	**400**	400	**400**	400	**400**	400	400
6	**323**	323	**323**	323	**323**	323	323
7	672	692	**665**	692	**665**	679	665
8	**522**	522	**522**	522	**522**	522	522
9	**522**	629	**522**	536	**522**	536	522
10	**453**	636	**453**	636	**453**	636	453
11	433	433	**423**	453	**423**	433	423
12	**549**	626	**549**	616	**549**	603	549
13	659	659	633	659	**626**	659	[596]
14	**503**	1009	633	905	**503**	633	519
15	756	962	782	962	**709**	962	736
16	906	1438	872	1432	**653**	1292	683
17	1658	2454	1462	2623	**1324**	2444	[1189]
18	1547	2304	1801	2300	**1152**	2114	1312
19	1968	2740	1700	2767	**1691**	2710	1698
20	1780	2900	**1628**	2860	**1628**	2860	1898

TAB. 6.3 – Meilleures valeurs obtenues par Divide-and-Evolve après le *racing* et LPG pour chaque instance du domaine `zeno-simple-time` pour chaque hypothèse de régularité. En gras les meilleurs valeurs trouvées par Divide-and-Evolve. Dans la dernière colonne les valeurs trouvées par LPG. Surlignées les valeurs de référence inférieures à celles trouvées par Divide-and-Evolve et les valeurs entre [] indique que Divide-and-Evolve n'a pas égalé la solution de référence. Pour la plupart des grandes instances, Divide-and-Evolve est meilleur que LPG.

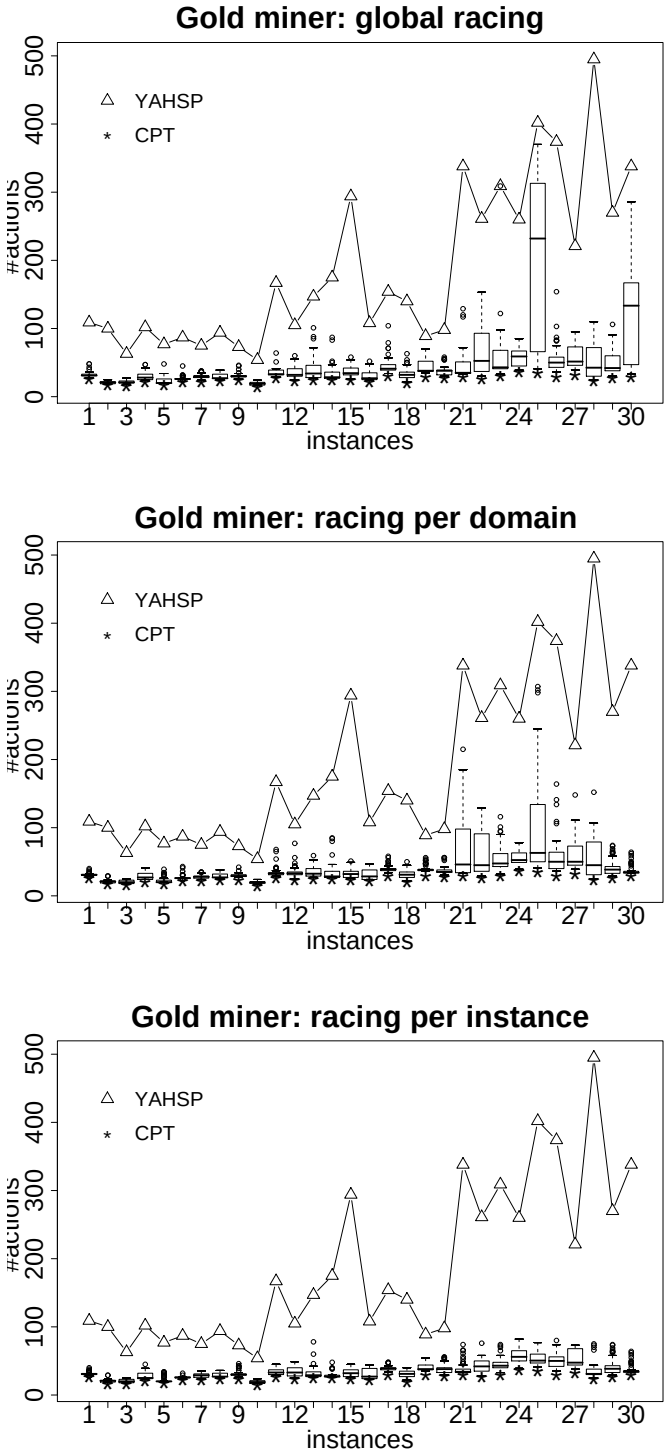

FIG. 6.4 – Les Box-plots représentent les résultats de Divide-and-Evolve sur le domaine `gold-miner` après le *racing*, pour chaque hypothèse de régularité. Pour chaque instance, △ (respectivement *) représente le nombre d'actions trouvé par YAHSP (respectivement trouvé par CPT).

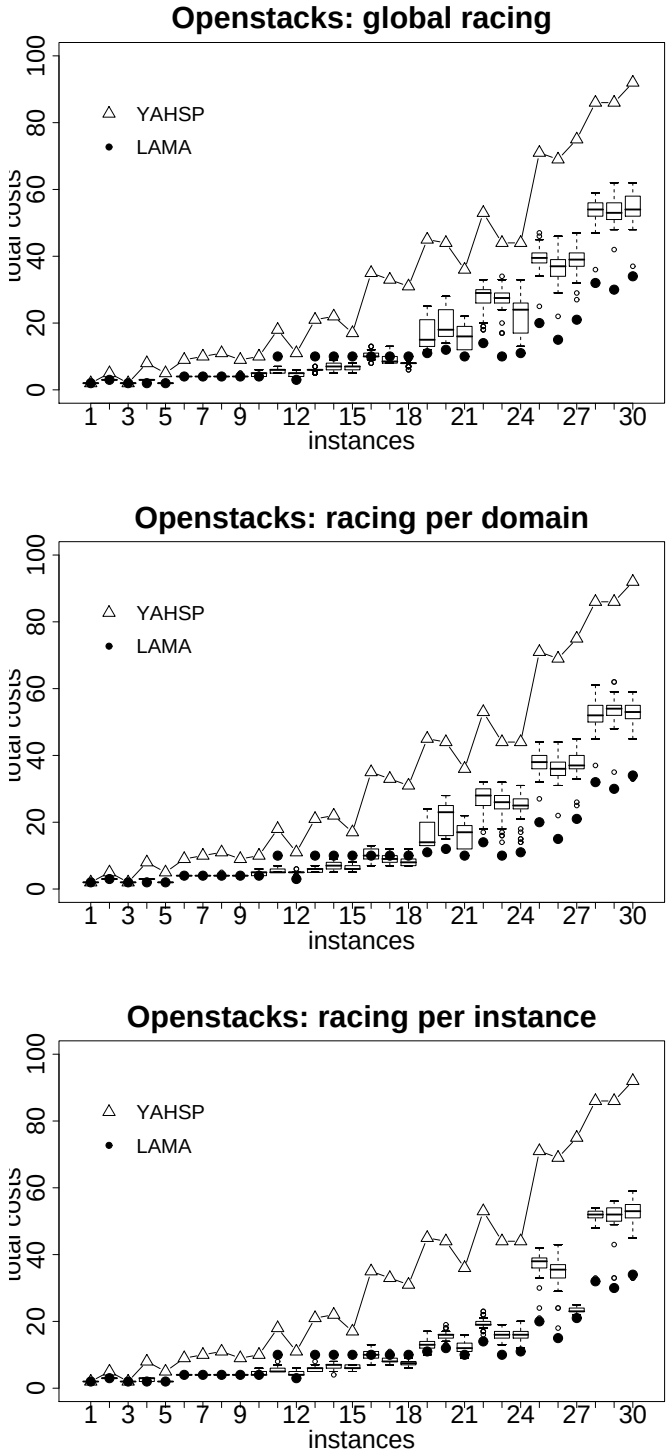

FIG. 6.5 – Les Box-plots représentent les résultats de Divide-and-Evolve sur le domaine `openstacks-ipc6-satisficing-track` après le *racing*, pour chaque hypothèse de régularité. Pour chaque instance, △ (respectivement •) représente le coût total des actions trouvé par YAHSP (respectivement trouvé par LAMA).

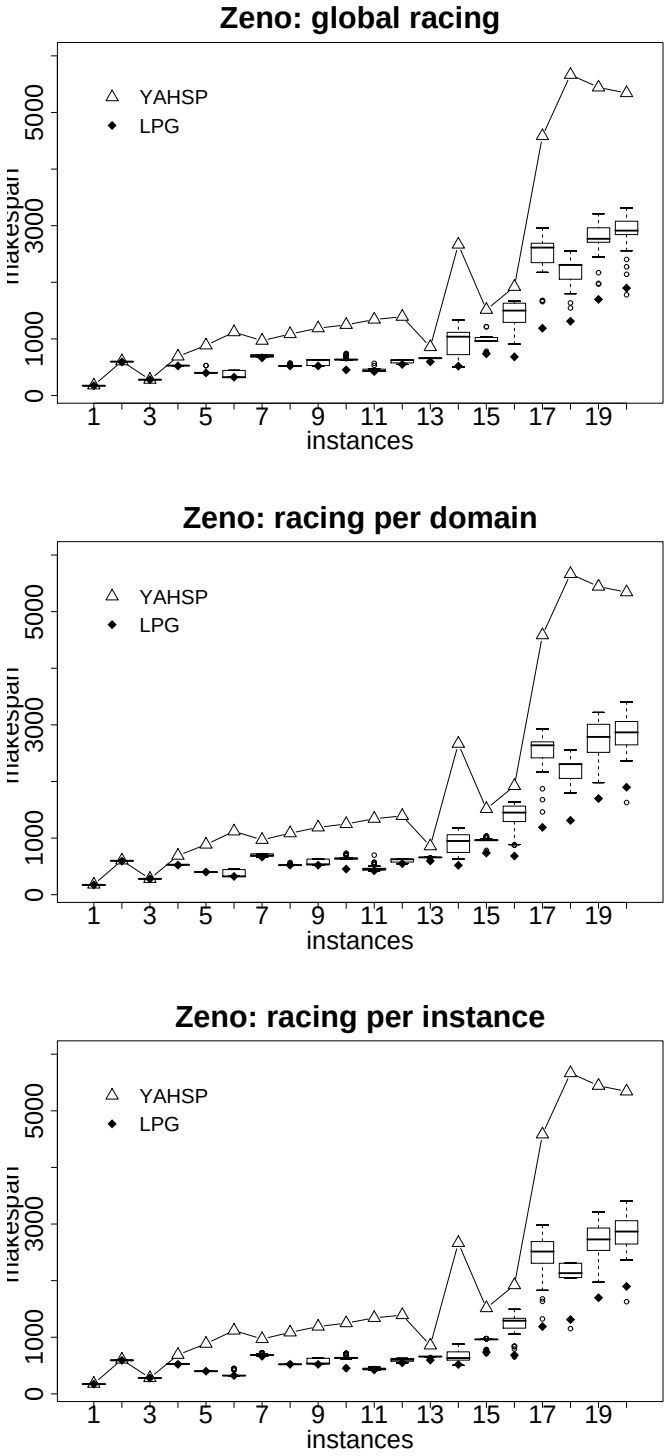

FIG. 6.6 – Les Box-plots représentent les résultats de Divide-and-Evolve sur le domaine `zeno-simple-time` après le *racing*, pour chaque hypothèse de régularité. Pour chaque instance, △ (respectivement ♦) représente le makespan trouvé par YAHSP (respectivement trouvé par LPG).

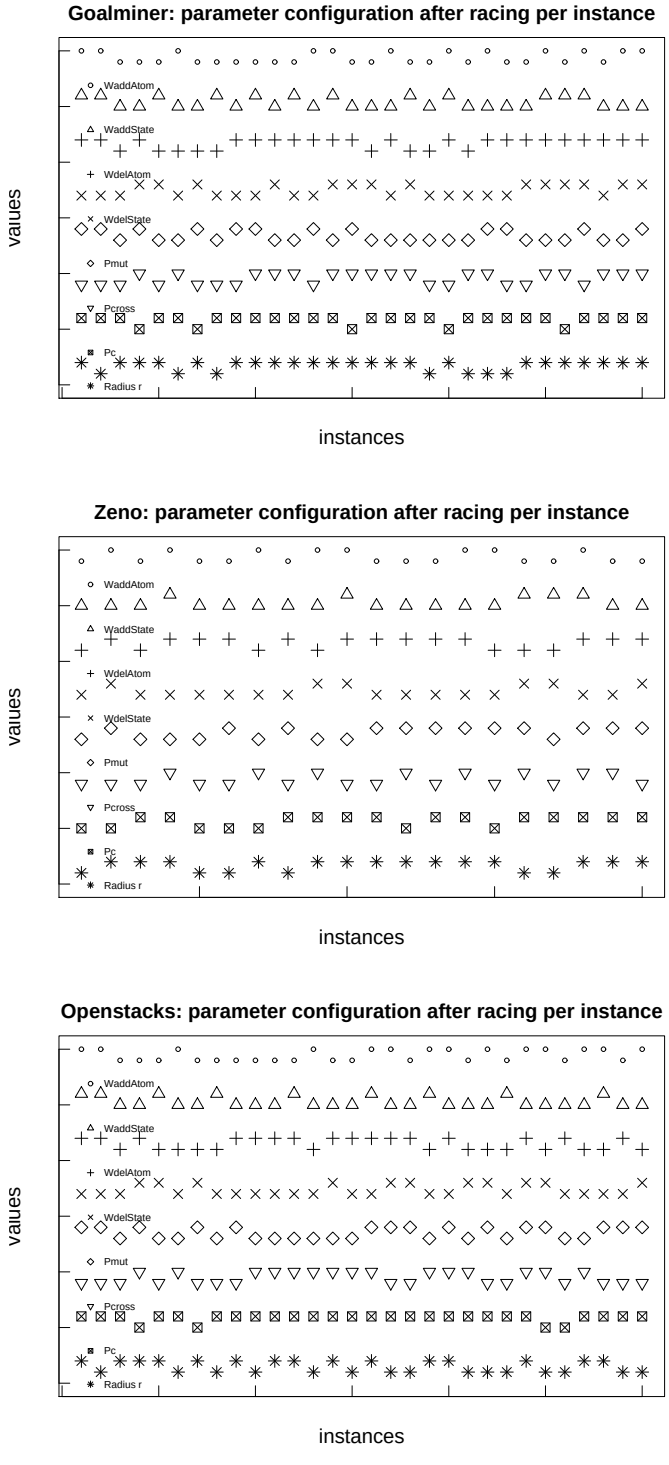

FIG. 6.7 – Meilleures configurations de paramètres, pour chaque instance des domaines `gold-miner`, `zeno simple time` et `openstacks-ipc6-satisficing-track`. De haut en bas les paramètres, $w_{addAtom}$, $w_{addState}$, $w_{delAtom}$, $w_{delState}$, p_{mut}, p_{cross}, p_c, r. Dans le plan d'expérience utilisé pour ces résultats, chaque paramètre a deux valeurs possible. Le *racing* par domaine est la dernière colonne de chaque figure.

Chapitre 7

Résultats expérimentaux

Sommaire

7.1 Conditions expérimentales

Les systèmes Divide-and-Evolve ont été implantés en C++ à partir du framework évolutionnaire EO[1]. Le planificateur approché YAHSP [Vidal 2004b] et le planificateur optimal CPT [Vidal 2004a] ont été embarqués dans les différentes implantations de Divide-and-Evolve afin de tester l'impact du choix d'un planificateur optimal ou non. La comparaison de ces différentes implantations a été ensuite faite avec les meilleurs systèmes actuels (mis à jour) pour chaque type de problème de planification. Les tests ont quant à eux été effectués sur des machines Linux de 2GHz de fréquence d'horloge, 6Mo de mémoire cache et 16 Go de mémoire vive (ou RAM) sur les problèmes des benchmarks de la compétition internationale de planification (IPC).

[1] http ://eodev.sourceforge.net/

7.1.1 Mesures de performance

Les mesures utilisées lors de nos évaluations sont celles de la compétition IPC. Il est à noter que lors de la compétition IPC, le temps d'exécution des planificateurs est limité à **30 minutes**.

La **mesure empirique de la qualité** d'une solution produite par un planificateur **sur une instance** i d'un domaine donné est définie par le rapport Q_i^*/Q_i, Q_i^* étant la valeur de référence de la qualité de la solution de l'instance i et Q_i la qualité de la solution produite par le planificateur. Si le planificateur ne trouve pas de solution, alors le rapport Q_i^*/Q_i sera artificiellement mis à 0.

La **mesure empirique de la qualité** des solutions d'un planificateur **sur un domaine** est la somme sur toutes les instances des mesures empiriques de la qualité. La mesure empirique de la qualité des solutions d'un planificateur sur un ensemble de domaines est la somme sur tous les domaines des mesures empiriques de qualité du planificateur. Ainsi, le planificateur ayant obtenu la plus grande mesure empirique sera considéré comme meilleur planificateur de l'instance, du domaine ou des domaines.

Cependant, pour les planificateurs stochastiques comme par exemple les systèmes Divide-and-Evolve, aucune conclusion ne peut être tirée après une seule exécution de l'algorithme. Dans le but d'évaluer la robustesse de ces planificateurs, nous avons réalisé 11 exécutions indépendantes de ces planificateurs sur chaque instance. Pour ces systèmes, le nombre d'instances résolues (ou **coverage**) sera donc le nombre total d'instances résolues au moins une fois (ou au moins six fois). La qualité de la solution d'une instance d'un domaine donné prise en compte pour le calcul de la **mesure empirique de la qualité** sera la meilleure des 11 valeurs potentielles trouvées par le planificateur stochastique.

La moyenne des succès d'un planificateur stochastique sur un domaine $\mathcal{D}$ donné (ou **average coverage**) sera définie par :

$$\textbf{average coverage} = \frac{\sum_{i;n_i>0} n_i}{\sum_{i;n_i>0} 1} \tag{7.1}$$

où n_i est le nombre de succès de l'algorithme lors de la résolution de l'instance i de $\mathcal{D}$. Cette moyenne est donc comprise dans l'intervalle $[0, 11]$, et plus elle est proche 11, meilleure elle sera.

Enfin, la moyenne des mesures de qualité (ou **average quality**) des planificateurs stochastiques sur un domaine $\mathcal{D}$ donné est la somme de la somme des inverses moyennes harmoniques des solutions obtenues par ces planificateurs pondérées par les valeurs de référence. En d'autres termes :

$$\textbf{average quality} = \sum_{instances\ i\ of\ \mathcal{D}} \frac{1}{n_i} \sum_{\{\text{run } j \text{ solved } i\}} \frac{Q_i^*}{q_j} \tag{7.2}$$

où i désigne une instance du domaine $\mathcal{D}$, q_j la valeur de la qualité de la solution trouvée au cours de l'exécution j sur le problème i. Cette moyenne est donc comprise dans $[1, \#instances]$, avec $\#instances$ le nombre d'instances du domaine. Plus elle

sera proche de *#instances*, meilleure elle sera. Mais cette mesure cache les échecs de l'algorithme, elle doit donc être comparée au coverage.

7.2 Exemples de décompositions

La figure 7.1 montre deux décompositions permettant de guider le planificateur non-optimal YAHSP vers une solution optimale. Remarquons que les deux séries d'états trouvées par Divide-and-Evolve sont différentes.

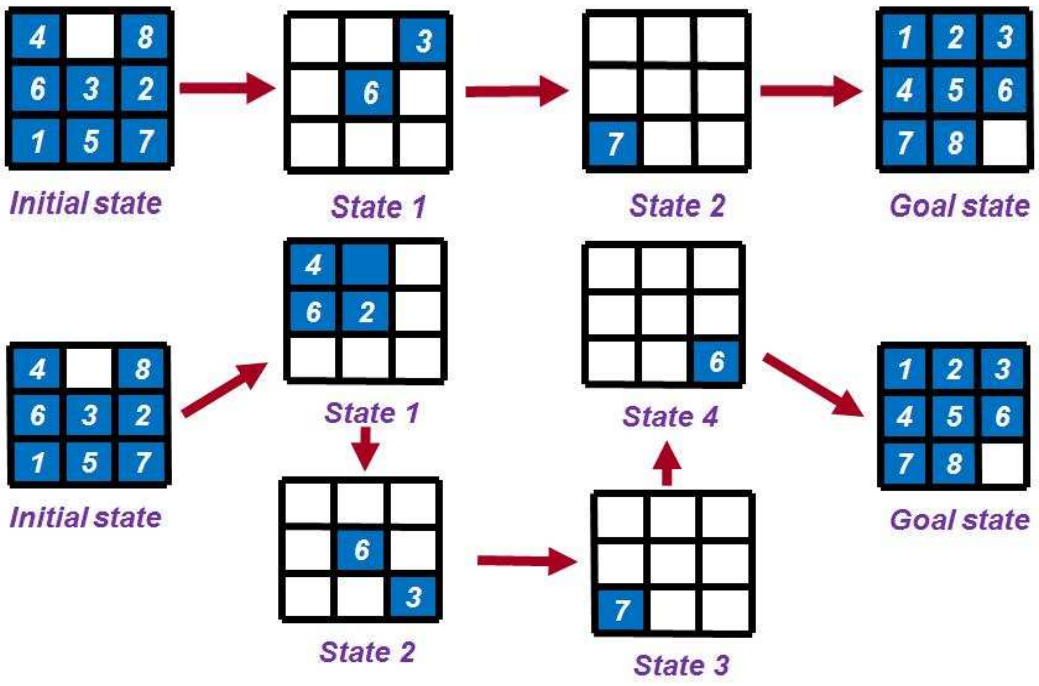

FIG. 7.1 – Jeu du taquin : 2 exemples de décompositions permettant au planificateur non-optimal YAHSP de résoudre le problème de façon optimal en 25 actions (la valeur optimal est calculée avec CPT) contre 49 actions pour YAHSP tout seul. La case sans chiffre représente la case ne contenant pas de tuille.

7.3 Comparaison des approches de construction des séries d'états

Dans cette section nous avons utilisé pour l'approche de construction aveugle des série d'états, les deux versions des systèmes Divide-and-Evolve qui ont participé à la dernière compétition IPC [Bibai 2008a] [2] implantant les algorithmes de la section 5.3 du chapitre 5 et la version décrite dans [Bibai 2010a] pour l'approche orientée

[2] http ://ipc.informatik.uni-freiburg.de/Planners

TAB. 7.1 – Configuration des paramètres de l'approche de construction orientée (haut) et l'approche de construction avaugle (bas).

	Parameters								
	P_{cross}	P_{mut}	$w_{addState}$	$w_{delState}$	$w_{addAtom}$	$w_{delAtom}$	P_c	P_a	r
Values	0.2	0.8	3	1	1	1	0.8	0.5	2
	P_{cross}	P_{mut}	$w_{addState}$	$w_{delState}$	$w_{addAtom}$	$w_{delAtom}$	P_c	P_a	$\#atomes$
Values	0.3	0.8	35	3	35	7	0.8	0.5	6

de construction des séries d'états implantant les algorithmes de la section 5.4 du chapitre 5.

7.3.1 Les configurations de paramètres

La configuration de paramètres utilisée ici est celle issue du racing global (cf. chapitre 7.7) sur les domaines `gold-miner` (planification STRIPS classique), `zeno simple time` (planification temporelle) et `openstacks-ipc6-satisficing-track` (planification avec coûts) des benchmarks de la compétition internationale de planification (IPC). La configuration de paramètres retenue pour l'approche orientée est la suivante (cf. table 7.1) : les 3 probabilités $(p_{\text{cross}}, p_{\text{mut}}, p_c) = (0.2, 0.8, 0.8)$, les 4 coefficients de proportionnalité associés à chaque opérateur de mutation (w_{addState}, w_{delState}, w_{addAtom}, w_{delAtom}) $= (3, 1, 1, 1)$, le rayon de voisinage $r = 2$ et l'heuristique h^1[Haslum 2000] pour le calcul des dates au plus tôt des atomes. La configuration de paramètres retenue pour l'approche aveugle de construction des séries d'états est la suivante : les 3 probabilités $(p_{\text{cross}}, p_{\text{mut}}, p_c) = (0.3, 0.8, 0.8)$, les 4 coefficients de proportionnalité associés à chaque opérateur de mutation (w_{addState}, w_{delState}, w_{addAtom}, w_{delAtom}) $= (35, 3, 35, 7)$, et la taille maximale d'un état $\#atomes = 6$.

Comme l'approche de construction aveugle des séries d'états a été historiquement introduite pour la résolution des problèmes de planification temporelle, les domaines choisis ici pour la comparaison des différentes approches de construction des séries d'états sont ceux de la planification temporelle pour lesquels cette approche permet d'obtenir de bons résultats.

7.3.2 Résultats

Nous nous positionnons ici dans la cadre de la conception. En d'autres termes, seule la qualité des solutions trouvées par chaque implantation nous intéresse. La durée maximale a été fixée à 2 heures pour chaque exécution de l'algorithme pour les variantes utilisant le planificateur CPT et de 30 minutes pour ceux utilisant le planificateur YAHSP. Les tables 7.2 et 7.3 présentent les meilleurs résultats obtenus par les différentes implantations de Divide-and-Evolve après 11 exécutions de cha-

cune d'elles sur une instance donnée. Ces résultats ont été obtenus en embarquant d'une part le planificateur optimal CPT [Vidal 2004a] et d'autre part le planificateur non-optimal YAHSP [Vidal 2004b]. DAE1 (respectivement DAE2) est la version implantant l'approche aveugle avec l'ensemble des prédicats restreint à ceux du but (respectivement à l'ensemble des prédicats restreint à la règle empirique basée sur la proportion des atomes - cf. chapitre 5 section 5.3.1). DAEx est la version implantant l'approche orientée de construction des séries d'états (cf. chapitre 5 section 5.4).

Les figures 7.2, 7.3, 7.4 et 7.5 rendent compte de la diversité des solutions obtenues par les différentes implantations utilisant le planificateur optimal CPT.

TAB. 7.2 – Divide-and-Evolve + planificateur embarqué CPT : meilleures valeurs trouvées par les différentes implantations des approches de construction des séries d'états de Divide-and-Evolve après 11 exécutions limité à 2 heures maximum de chaque version sur une instance d'un domaine donné. DAE1 (respectivement DAE2) est la version implantant l'approche aveugle avec l'ensemble des prédicats restreint à ceux du but (respectivement à l'ensemble des prédicats restreint à la règle empirique basée sur la proportion des atomes - cf. 5.3.1). DAEx est la version implémentant l'approche orientée.

Planners	Zeno simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**173**	**592**	**280**	**522**	**400**	**323**	**665**	**522**	**522**	**453**	**423**	**549**	**596**	576	1298	-	-	-	-	-
DAE1	**173**	**592**	**280**	**522**	**400**	**323**	**665**	**522**	536	**453**	**423**	**549**	**596**	503	1035	**699**	-	-	-	-
DAE2	**173**	**592**	**280**	**522**	**400**	**323**	**665**	**522**	**522**	**453**	**423**	**549**	**596**	**476**	**825**	-	-	-	-	-

Planners	Satellite simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**46**	**70**	**34**	**58**	**36**	**46**	**34**	**46**	**34**	**43**	**46**	**79**	-	**48**	**51**	-	**34**	**46**	-	-
DAE1	**46**	**70**	**34**	**58**	**36**	**46**	**34**	51	39	56	**46**	92	**97**	58	68	**39**	**34**	56	-	-
DAE2	**46**	**70**	**34**	**58**	**36**	**46**	**34**	56	**34**	**43**	**46**	118	-	58	61	-	**34**	**46**	-	-

Planners	Rovers simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**53**	**43**	**53**	**45**	**93**	**125**	**73**	**105**	**103**	-	**105**	**88**	-	**107**	**122**	-	-	-	-	-
DAE1	**53**	**43**	**53**	**45**	**93**	141	**73**	**105**	105	-	**105**	**88**	-	**107**	125	**150**	-	-	-	-
DAE2	**53**	**43**	**53**	**45**	97	135	**73**	-	105	-	-	90	-	107	-	-	-	-	-	-

Planners	Driver simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**91**	**65**	**40**	**52**	**51**	**52**	**40**	**52**	**92**	**38**	**65**	-	**104**	-	**145**	-	-	-	-	-
DAE1	**91**	**65**	**40**	**52**	**51**	**52**	**40**	**52**	99	48	**65**	-	-	-	-	-	-	-	-	-
DAE2	**91**	**65**	**40**	**52**	**51**	**52**	**40**	59	**92**	**38**	74	-	-	-	-	-	-	-	-	-

TAB. 7.3 – Divide-and-Evolve + planificateur embarqué YAHSP : meilleures valeurs trouvées par les différentes implantations des approches de construction des séries d'états de Divide-and-Evolve après 11 exécutions limité à 30 minutes maximum de chaque version sur une instance d'un domaine donné. DAE1 (respectivement DAE2) est la version implantant l'approche aveugle avec l'ensemble des prédicats restreint à ceux du but (respectivement à l'ensemble des prédicats restreint à la règle empirique basée sur la proportion des atomes - cf. 5.3.1). DAEx est la version implantant l'approche orientée.

Planners	Zeno simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**173**	**592**	**280**	**529**	**400**	**323**	**672**	**522**	**522**	**513**	**423**	**549**	**622**	**569**	**709**	**779**	**1550**	**1645**	**1691**	**1628**
DAE1	180	606	**280**	536	536	380	706	536	536	713	473	583	856	816	969	916	1848	-	-	-
DAE2	**173**	606	**280**	**529**	522	480	716	782	683	743	483	626	866	1039	1009	1338	2221	2101	2231	2900

Planners	Satellite simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**46**	**70**	**34**	**58**	**36**	**46**	**34**	**46**	**39**	**46**	**46**	**91**	**80**	**46**	**56**	**46**	**44**	**46**	**87**	**106**
DAE1	**46**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DAE2	**46**	90	-	76	-	-	-	-	-	-	-	-	-	-	-	-	-	-	144	236

Planners	Rovers simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**53**	**43**	**53**	**45**	**93**	**130**	**73**	**105**	**103**	**133**	**105**	**88**	**143**	**107**	**122**	**138**	**168**	**140**	**222**	**237**
DAE1	**53**	**43**	67	**45**	-	-	**73**	-	105	-	108	-	143	-	128	141	-	-	-	-
DAE2	**53**	45	67	**45**	-	-	77	-	136	-	139	-	-	-	133	142	170	143	-	-

Planners	Driver simple time domain																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
DAEx	**91**	**65**	**40**	**52**	**57**	**52**	**40**	**53**	**92**	**49**	**65**	**170**	**124**	**126**	**123**	-	**239**	**360**	**666**	**382**
DAE1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DAE2	**91**	152	**40**	77	**57**	75	61	103	120	**49**	139	378	214	189	194	-	419	538	1010	442

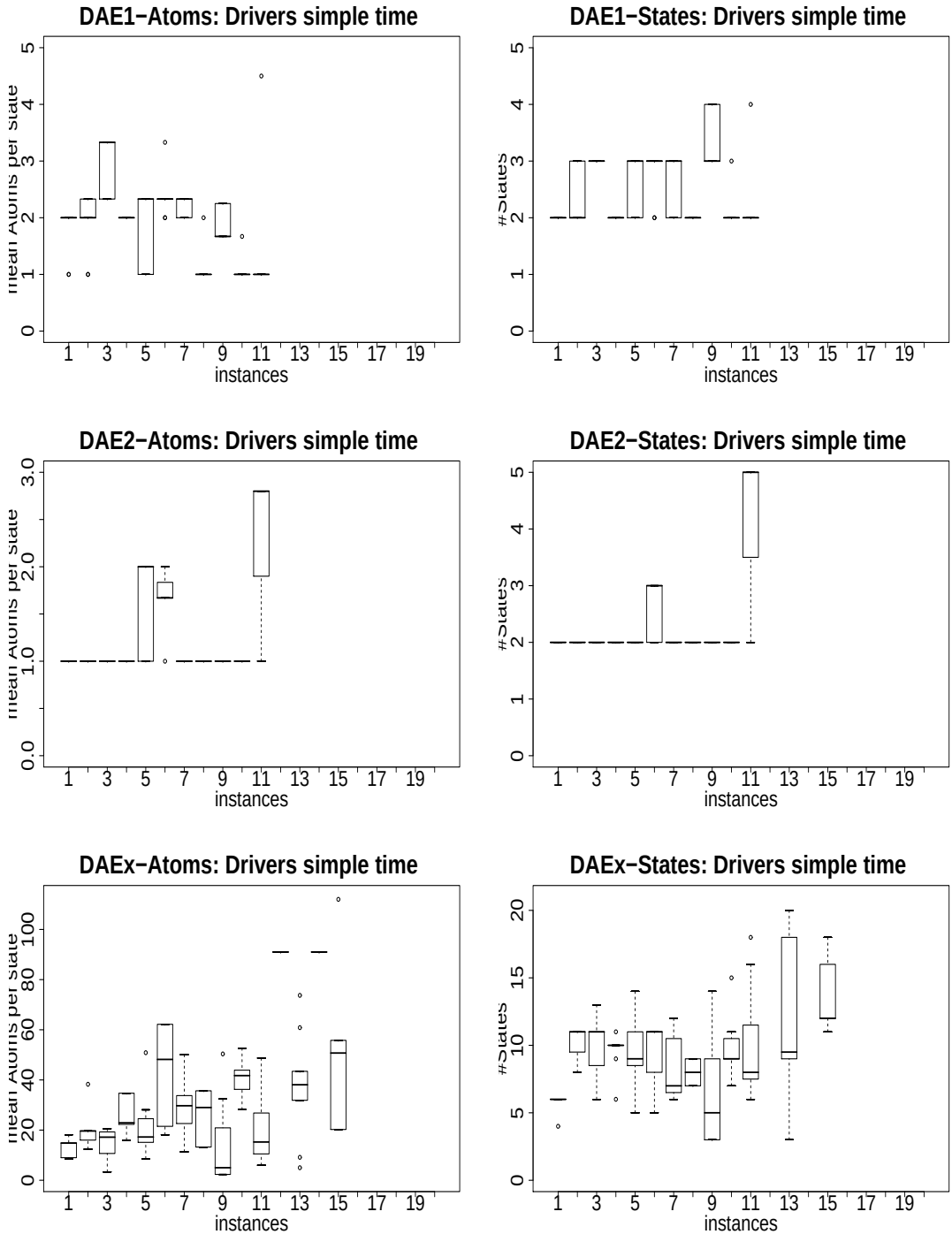

FIG. 7.2 – Tailles des séries d'états et tailles moyennes des états de chaque variante de Divide-and-Evolve sur le domaine `drivers simple time`.

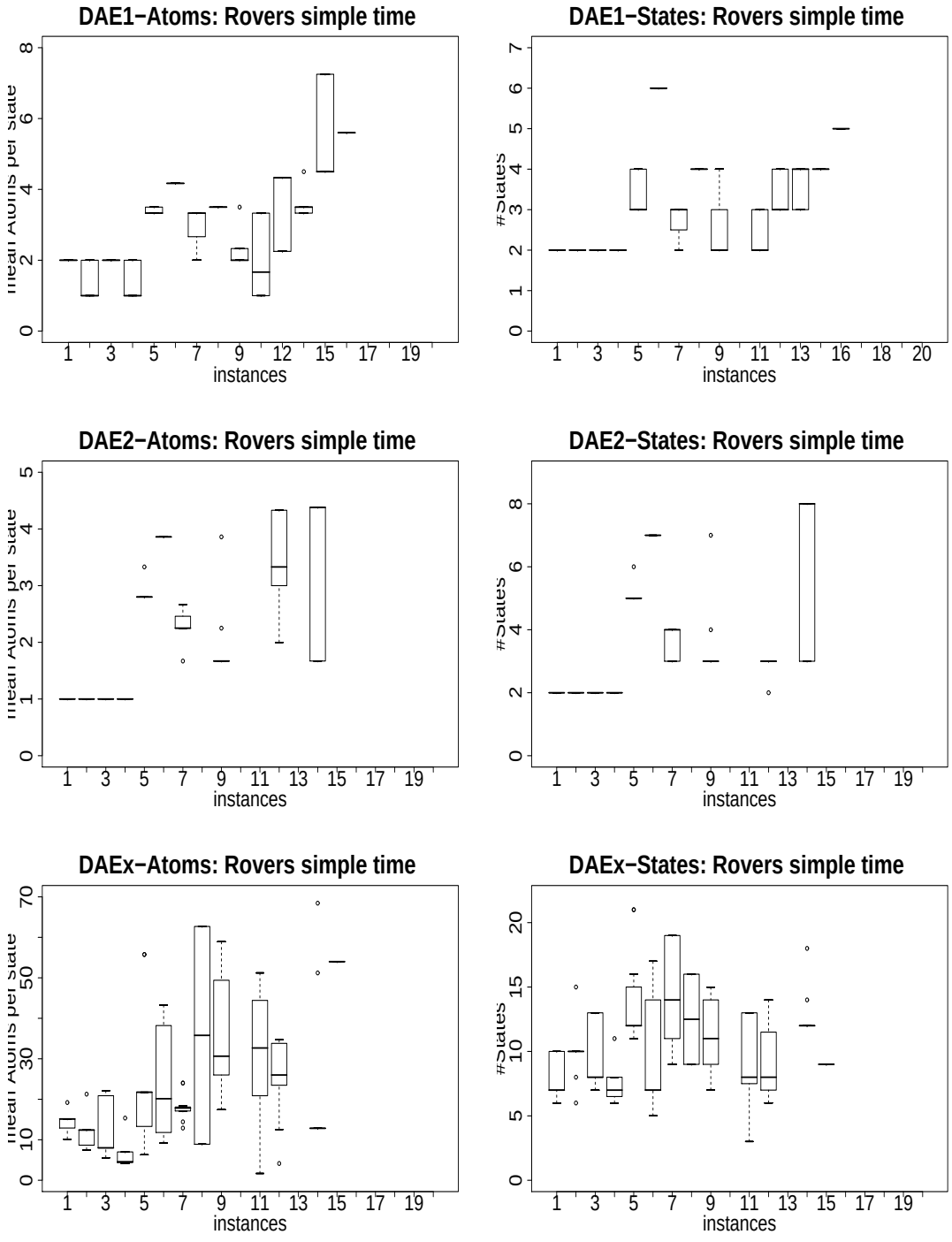

FIG. 7.3 – Tailles des séries d'états et tailles moyennes des états de chaque variante de Divide-and-Evolve sur le domaine `rovers simple time`.

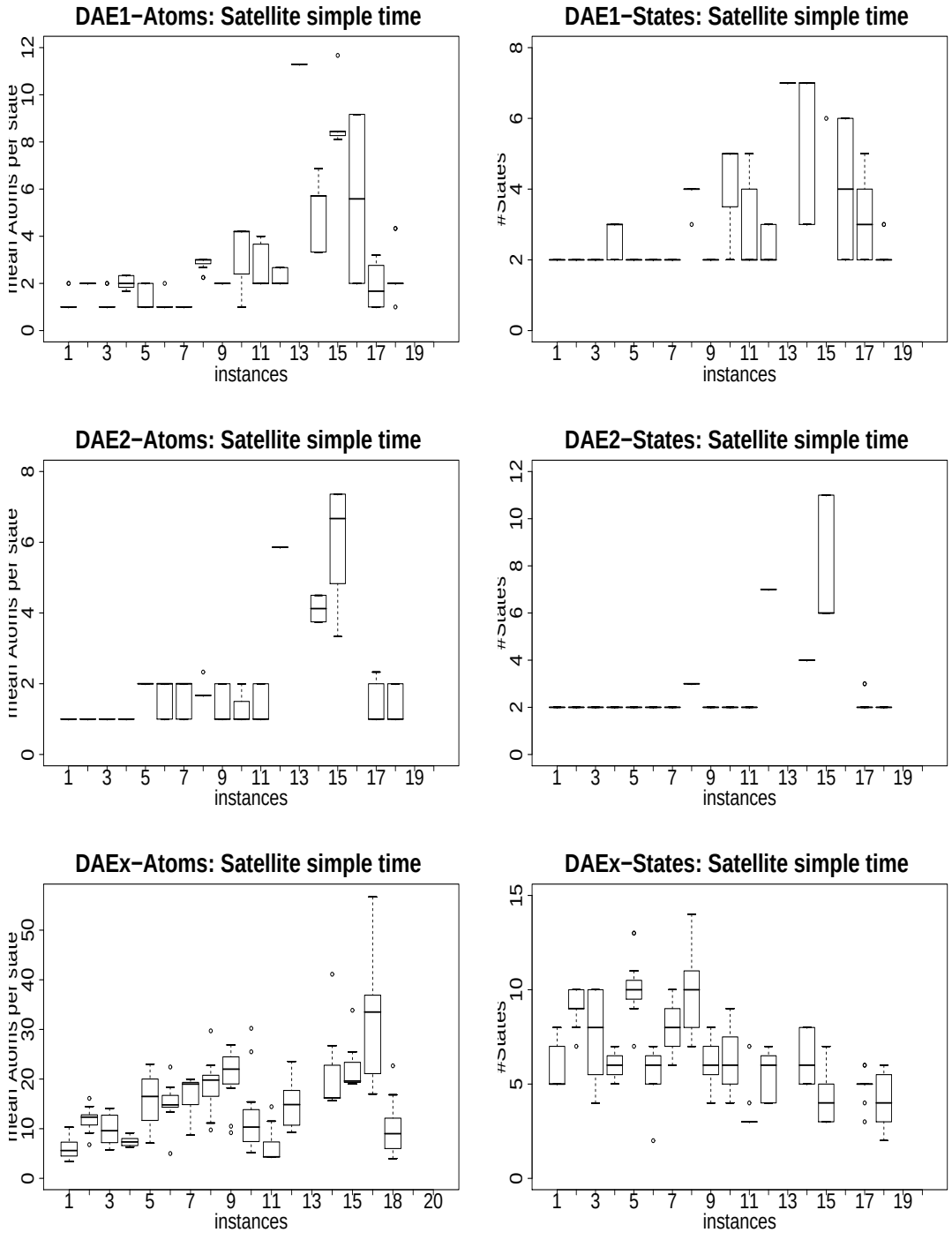

FIG. 7.4 – Tailles des séries d'états et tailles moyennes des états de chaque variante de Divide-and-Evolve sur le domaine `satellite simple time`.

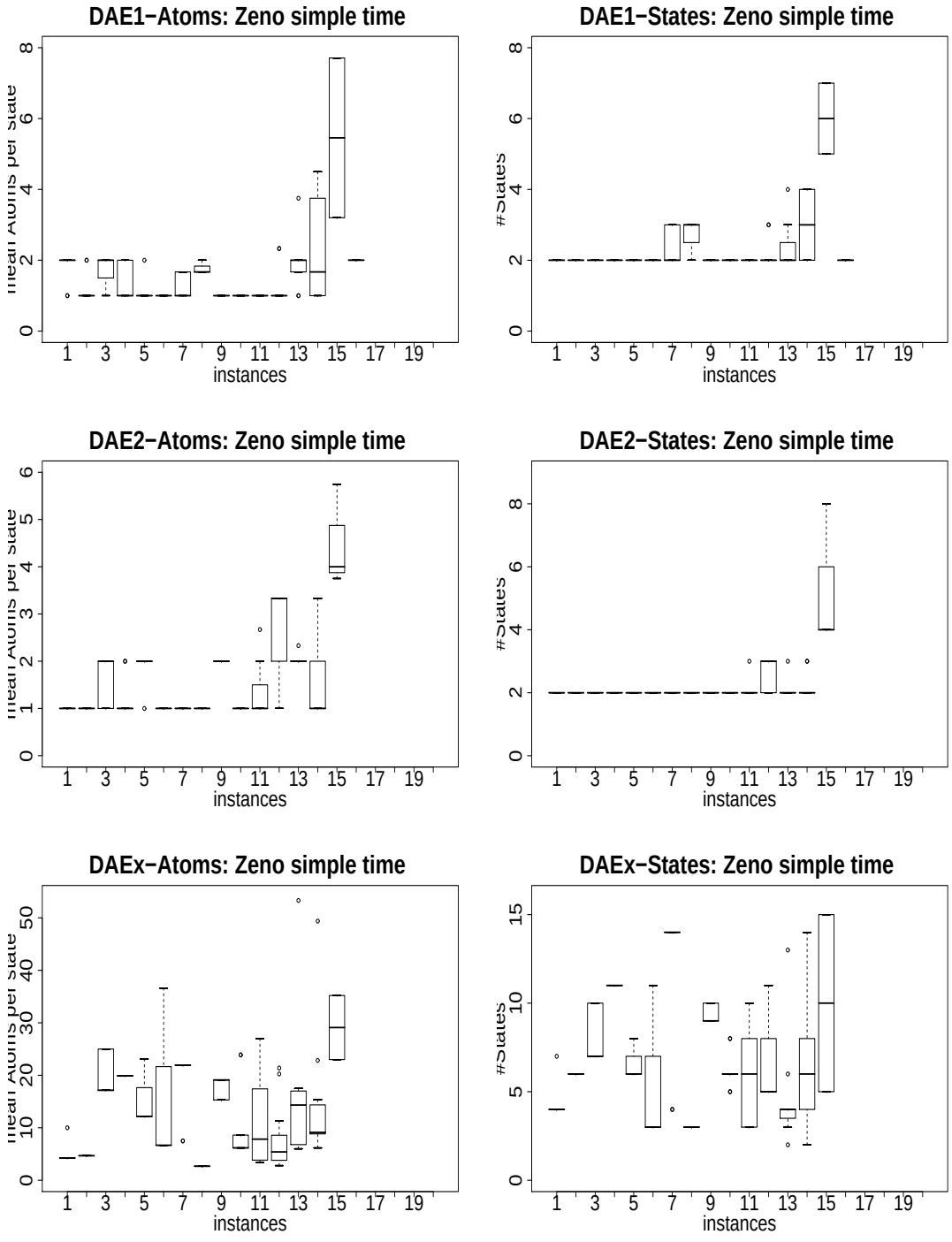

FIG. 7.5 – Tailles des séries d'états et tailles moyennes des états de chaque variante de Divide-and-Evolve sur le domaine `zeno simple time`.

7.3.3 Discussions

Significativement, il y a plus de diversités en tailles des séries d'états (respectivement en tailles des états) dans les solutions trouvées par DAEx par rapport à celles trouvées par DAE1 et DAE2. Pour les petites instances, par exemple les instances 1 à 4 du domaine `rovers simple time`, il n'y a pratiquement aucune diversité dans la tailles des séries d'états des solutions trouvées par DAE1 et DAE2 alors que celles de DAEx varient.

En ce qui concerne la qualité des solutions obtenues, pour les variantes utilisant le planificateur optimal CPT, alors que la qualité des solutions des deux approches de construction aveugle des séries d'états varie en fonction des instances, DAEx trouve toujours (excepté faute de temps pour les instances 14 et 15 du domaine `zeno simple time` - cf. table 7.2) la meilleure valeur. Cette dépendance entre les prédicats et la qualité des solutions est amplifiée lorsque Divide-and-Evolve embarque le planificateur non-optimal YAHSP (cf. table 7.3). Pour ces variantes (utilisant le planificateur YAHSP), on remarque que la qualité des solutions trouvées par DAEx sest toujours la meilleure comparée à celle de DAE1 et DAE2 (cf. table 7.3). Le même comportement a aussi été observé sur les benchmarks temporels de la dernière compétition IPC [Bibai 2009c]. On peut donc en déduire empiriquement que la qualité des solutions dépend des prédicats choisis pour la construction des séries d'états (voir par exemple les instances 8 et 9 du domaine `drivers simple time` table 7.2). Remarquons que la variante de DAEx embarquant le planificateur YAHSP est la seule inplantation qui résout tous les problèmes (excepté l'instance 16 du domaine `drivers simple time`) en 30 minutes (cf. table 7.3).

Il apparaît donc clairement que DAEx est globalement plus robuste que DAE1 et DAE2, par conséquent, pour la suite de ce chapitre, nous ne présenterons que les résultats obtenus avec DAEx.

7.4 Impact du choix d'un planificateur embarqué

Les résultats de cette section ont été publiés à la conférence *Evolutionary Computation in Combinatorial Optimization* (EvoCOP) [Bibai 2010b].

Dans cette section, nous analysons les conséquences du choix d'un planificateur embarqué optimal contre le choix d'un planificateur embarqué non optimal sur la qualité des solutions et durée d'exécution de DAEx. Pour cela, nous avons embarqué les planificateurs optimal CPT [Vidal 2004a] et approché YAHSP[Vidal 2004b] dans DAEx. Les paramètres de DAEx sont les mêmes que ceux de la section 7.3.1.

Afin d'analyser le comportement des différentes variantes de DAEx sur les différents types de problèmes de planification, nous avons choisi d'effectuer nos expériences sur les domaines suivants : `airport`, `satellite` et `logistics` pour les problèmes de planification STRIPS classique, `openstacks`, `scanalyser` et `woodworking` de la sixième compétition internationale de planification (*sequential satisficing track*) pour les problèmes de planification avec coûts, et `crewplanning`, `elevator` et `satellite time windows compiled` pour les problèmes de planification temporelle.

Pour les résultats ci-dessous, la durée maximale d'une exécution d'une variante de DAEx a été fixée à nouveau à 2 heures.

7.4.1 Résultats

Les tables 7.4 présentent pour chaque domaine et type de problème, la mesure empirique de qualité ($Q_{planner}$), le nombre de problèmes résolus ($S_{planner}$) et la table 7.5 le ratio $Q_{planner}/S_{planner}$ des planificateurs embarqués YAHSP, CPT, et les deux variantes de DAEx. Pour les deux variantes de DAEx, les valeurs entre parenthèses sont les moyennes des mesures de qualité et les moyennes de succès de chacune de ces variantes sur un domaine donné.

La figure 7.6 p.108 montre la distribution des solutions des deux variantes de DAEx, et les valeurs des solutions obtenues avec les différents planificateurs embarqués. Chaque colonne représente une instance du domaine. Pour les planificateurs YAHSP et CPT, les symboles ('△' et '¤' respectivement) indiquent la qualité des solutions trouvées. La figure 7.7 montre les distributions des durées d'exécution des deux variantes de DAEx sur le domaine `crewplanning`. La figure 7.8 montre le comportement de la fitness des deux variantes de DAEx sur l'instance 2 du domaine `elevator`.

7.4.2 Discussions

Premièrement, la variante DAEx+YAHSP résout de manière significative plus d'instances (79.36% des instances testées) que le planificateur embarqué YAHSP tout seul (71.49% des instances testés), et beaucoup plus que la variante DAEx+CPT (21.70% des instances testées) et le planificateur embarqué CPT tout seul (10% seulement). La variante DAEx+YAHSP a aussi la meilleure mesure de qualité (cf. tables 7.4 -tableau 2) sur tous les types de problèmes de planification testés. En plus, la variante DAEx+YAHSP trouve le plus souvent (cf. table 7.5) la valeur optimale (calculée par le planificateur optimal CPT), ou une valeur proche à 94% (rapport entre la valeur optimale et la valeur trouvée) de l'optimum (cf. figure 7.6), et trouve toujours une solution de meilleure qualité que le planificateur YAHSP (cf. la table 7.4 et la figure 7.6). Néanmoins, pour toutes les instances, par exemple instance 2 du domaine `elevator` (cf. figure 7.8), les fitness des deux variantes de DAEx améliorent graduellement la qualité des solutions des problèmes résolus. Les durées d'exécution de la variante DAEx+YAHSP, par exemple sur le domaine `crewplanning` (cf. figure 7.7), sont toujours inférieures à celles de la variante DAEx+CPT (en effet, 2 heures ne suffisent souvent pas pour la variante DAEx+CPT). Par conséquent, la variance des qualités des solutions (cf. la figure 7.6) produites par la variante DAEx+YAHSP est toujours inférieure à celle de la variante DAEx+CPT même lorsque DAEx+CPT trouve une *bonne* solution.

Cependant, bien que la variante DAEx+YAHSP a le meilleur ratio (pour les planificateurs non optimaux) sur tous les problèmes testés (cf. table 7.5), la variante DAEx+CPT a le meilleur ratio sur les problèmes de planification temporelle

TAB. 7.4 – Qualité et nombre de problèmes résolus du planificateur optimal CPT, et des planificateurs approchés YAHSP, DAEX+CPT et DAEX+YAHSP sur les domaines testés. Dans la colonne `Domain`(x), x est le nombre total d'instances d'un domaine. Les colonnes de la première table indiquent le nombre d'instances résolues (et pour les deux variantes de Divide-and-Evolve le nombre moyen de solutions trouvées sur 11 exécutions). Les colonnes du seconde table représentent la valeur de la mesure qualitative (et entre parenthèses, pour les variantes de Divide-and-Evolve la moyenne quadratique de cette mesure). Les valeurs en gras sont les meilleures valeurs obtenues pour chaque type de problème de planification testé (STRIPS, planification avec coûts, et planification temporelle).

Domain	Coverage			
	YAHSP	CPT	DAEX+YAHSP	DAEX+CPT
Airport-STRIPS(50)	46	7	43 / 8.2	22 / 7.2
Satellite-STRIPS(36)	24	2	31 / 10.4	4 / 11
Logistics-STRIPS(198)	125	9	142 / 9.3	11 / 9.9
Total problems (284)	*195*	*18*	***216***	*37*
Openstacks-Cost(30)	30	3	30 / 10.9	6 / 6.8
Scanalyser-Cost(30)	27	3	28 / 10.5	6 / 7.7
Woodworking-Cost(30)	23	1	23 / 10.4	5 / 9
Total problems (90)	*80*	*7*	***81***	*17*
Crewplanning-Time(30)	30	14	30 / 10.9	30 / 9.5
Elevator-Time(30)	21	1	30 / 10.8	4 / 7
Satellite-Time(36)	10	7	16 / 9.2	14 / 8.1
Total problems (96)	*61*	*22*	***76***	*48*
Total (470)	336	47	**373**	102

Domain	Quality			
	YAHSP	CPT	DAEX+YAHSP	DAEX+CPT
Airport-STRIPS(50)	44.34	7	42.62 (42.1)	22 (22)
Satellite-STRIPS(36)	14.08	2	31 (30.8)	3.80 (3)
Logistics-STRIPS(198)	92.07	9	141.94 (133.5)	10.93 (10.9)
Total quality (284)	*150.50*	*18*	***215.56***	*36.74*
Openstacks-Cost(30)	11.60	3	30 (27.2)	5 (4.1)
Scanalyser-Cost(30)	16.48	3	27.81 (26)	5.92 (5.8)
Woodworking-Cost(30)	20.10	1	22.98 (22.3)	5 (3)
Total quality (90)	*48.19*	*7*	*80.80*	*15.92*
Crewplanning-Time(30)	24.65	14	29.97 (29.5)	29.01 (27.4)
Elevator-Time(30)	11.55	1	28.99 (23.6)	3.86 (3.6)
Satellite-Time(36)	7.47	7	15.15 (14.5)	14 (13.7)
Total quality (96)	*43.69*	*22*	***74.12***	*46.88*
Total (470)	242.39	47	**370.49**	99.56

TAB. 7.5 – Ratios $\frac{Quality\ Score}{Coverage}$ des tables 7.5 pour chaque domaine ainsi que la moyenne de ces ratios pour tous les types de problèmes. Les valeurs soulignées sont les meilleures valeurs obtenues par les planificateurs approchés YAHSP, DAEX+CPT et DAEX+YAHSP sur chaque domaine.

Domain	Quality/ Total of solved problems			
	YAHSP	CPT	DAEX+YAHSP	DAEX+CPT
Airport-STRIPS(100%)	96.39%	100%	99.13%	100%
Satellite-STRIPS(100%)	58.69%	100%	100%	95.24%
Logistics-STRIPS(100%)	73.66%	100%	99.96%	99.45%
Total ratio (100%)	*76.25%*	***100%***	***99.70%***	*98.23%*
Openstacks-Cost(100%)	38.70%	100%	100%	83.33%
Scanalyser-Cost(100%)	61.04%	100%	99.35%	98.81%
Woodworking-Cost(100%)	87.42%	100%	99.95%	100%
Total ratio (100%)	*62.39%*	***100%***	***99.77%***	*94.05%*
Crewplanning-Time(100%)	82.19%	100%	99.93%	96.73%
Elevator-Time(100%)	55.03%	100%	96.64%	96.74%
Satellite-Time(100%)	74.78%	100%	94.72%	100%
Total ratio (100%)	*70.67%*	***100%***	*97.10%*	***97.82%***
Mean ratios (100%)	69.77%	**100%**	**98.85%**	96.70%

(cf. table 7.5). Ce qui pourrait s'expliquer par le parallélisme entre les actions et l'existence d'une routine plus élaborée de compression des solutions intermédiaires dans la variante DAEx+CPT, laquelle tire parti de la spécificité de la représentation en contrainte de son espace de recherche. Néanmoins, il n'existe pas une variante meilleure sur tous les types de domaines : même si une des deux variantes obtient le meilleur ratio sur un type de problème, il existe toujours au moins un domaine pour lequel l'autre variante obtient la meilleure qualité sur toutes ses instances résolues (cf. table 7.5). Voir par exemple, le domaine `crewplanning` pour les problèmes de planification temporelle, le domaine `airport` et `woodworking` pour les autres types de problèmes de planification.

Sur tous les domaines testés ici, la variante DAEx+CPT (respectivement la variante DAEx+YAHSP) résout plus d'instances que le planificateur embarqué CPT (respectivement YAHSP) tout seul. De plus, la moyenne de succès des deux variantes par domaine est très élevée (proche de la mesure maximale 11 - cf. équation 7.1). La variante DAEx+YAHSP semble être plus robuste que la variante DAEx+CPT : si une instance est résolue, alors presque toutes les exécutions de la variante DAEx+YAHSP mènent à une solution. En ce qui concerne la moyenne des mesures de qualité par domaine, la moyenne des deux variantes de DAEx est généralement supérieure à 90%.

Au vu de ces résultats, nous avons choisi d'utiliser la variante DAEx+YAHSP pour une comparaison avec l'état de l'art des planificateurs de la compétition in-

ternationale de planification IPC. Dans la suite, nous présenterons uniquement les résultats de la variante DAEx+YAHSP.

7.5 Impact du choix d'une heuristique

Dans cette partie, nous souhaitons comparer l'impact du choix de la fonction heuristique sur les résultats de DAEx+YAHSP ou de l'approche de construction orientée utilisant le planificateur embarqué YAHSP. Dans la version actuelle de cette implantation, il est possible d'utiliser deux heuristiques de la famille dite h^m[Haslum 2000] à savoir h^1 et h^2 (cf. chapitre 2 section 2.3.4). En fonction des paramètres obtenus après le *racing* global (cf. chapitre 7.7 section 6.3), nous allons tester le comportement de DAEx+YAHSP face à la modification de sa fonction heuristique.

7.5.1 Résultats

La figure 7.9 montre la distribution typique des qualités des solutions trouvées par DAEx+h^1 et DAEx+h^2 sur le domaine `rovers simple time`, et la figure 7.10 montre les superpositions des qualités des solutions trouvées par les deux variantes ainsi que la distribution des tailles des séries d'états et la distribution des tailles moyennes des états sur le même domaine.

7.5.2 Discussions

Clairement, bien que l'heuristique h^2 soit plus informative que l'heuristique h^1, il n'y a pas de différence significative entre les qualités des solutions trouvées par les deux variantes de DAEx testées ici (cf. figures 7.9 et 7.10). De même, il n'y a pas de différence significative sur la diversité des solutions trouvées par les deux variantes. La seule différence significative observée mais qui reste marginale porte sur les durées d'exécutions des deux variantes sur certaines instances du domaine testé. Par exemple les durées d'exécutions de chacune des variantes sur l'instance 16 (cf. figure 7.10). Le temps de calcul, plus ou moins important, de chacune de ces heuristiques pourrait expliquer ce comportement. Cependant, cette différence ne se répercute pas au niveau de la qualité des solutions trouvées ou même au niveau de la diversité de ces solutions (cf. figure 7.10).

Nous avons donc choisi la variante DAEx+YAHSP+h^1 pour une comparaison avec les meilleurs planificateurs actuels de la compétition IPC.

7.6 Comparaison avec les meilleurs planificateurs actuels

Les résultats de cette section ont été publiés à l'*International Conference on Automated Planning and Scheduling* (ICAPS) [Bibai 2010a].

Les expériences de ce chapitre ont été conduites d'après la règle de la compétition internationale de planification IPC sur tous les types de problèmes : planification

STRIPS classique, planification temporelle et planification avec coûts. Les benchmarks IPC suivant ont été choisis pour nos tests : les domaines `airport`, `freecell`, `openstacks`, `pathways`, `psr small`, `rovers`, `satellite` et `zeno` pour les problèmes de planification STRIPS classique, les domaines `openstacks`, `scanalyser`, `pegsolitaire`, `parcprinter`, `transport`, `elevator` et `woodworking` de la partie *satisficing track* de sixième compétition IPC pour les problèmes de planification avec coûts, et les domaines `crewplanning`, `elevator`, `openstacks`, `pegsolitaire`, `parcprinter`, `sokoban`, `rovers`, `zeno` et `satellite` pour les problèmes de planification temporelle. Soit un total global de 736 instances. La durée d'exécution maximale est de 30 minutes pour un planificateur sur une instance donnée (règle IPC).

Pour la sélection des domaines dans cette section, nous avons choisi pour les problèmes de planification temporelle et les problèmes de planification avec coûts, tous les domaines de la dernière compétition internationale de planification pouvant être résolus par le planificateur embarqué YAHSP ainsi que plusieurs domaines des compétitions précédentes pour lesquels nous avions des valeurs de référence. Les valeurs de références sont les meilleures valeurs des compétitions IPC, ou les meilleures valeurs obtenues avec DAE1 ou DAE2 et les valeurs optimales obtenues avec CPT. Pour les domaines de type STRIPS classique, les problèmes ont été choisit en fonction de leur complexité tel que définie dans [Helmert 2008], avec pour objectif d'avoir différents types de complexité.

En outre, les résultats de DaEx+YAHSP sont comparés pour chaque type de problème aux meilleurs planificateurs actuels : le planificateur LAMA [Richter 2008], version mise à jour (cf. annexe C), le planificateur LPG-td [Gerevini 2003a, Gerevini 2003b], et le planificateur TFD [Eyerich 2009], version mise à jour (cf. annexe B). Ce dernier, d'après les auteurs est le meilleur planificateur actuel de l'état de l'art des planificateurs temporelles. Bien sûr le planificateur embarqué YAHSP [Vidal 2004b] fait aussi partie de la comparaison. Les paramètres de chaque système sont ceux préconisés par les différents auteurs lors des compétitions IPC[3].

7.6.1 Résultats

Les tables 7.6, 7.9, 7.7, 7.10, 7.8 et 7.11 présentent pour chaque domaine et type de problème, pour le planificateur embarqué YAHSP, les planificateurs LAMA, LPG-td, TFD et DaEx+YAHSP, la mesure empirique de qualité ($Q_{planner}$) pour au moins un et six succès de DaEx+YAHSP, le nombre de problèmes résolus ($S_{planner}$) pour au moins un et six succès de DaEx+YAHSP et le ratio $Q_{planner}/S_{planner}$. Pour les planificateurs stochastiques DaEx+YAHSP et LPG-td, les valeurs entre parenthèses sont les moyennes des mesures de qualité et les moyennes de succès sur un domaine donné. Les résultats rapportés pour au moins six succès peuvent être vus comme la performance moyenne de l'algorithme, ceux avec un succès comme la performance *peak*.

[3] http ://idm-lab.org/wiki/icaps/index.php/Main/Competitions

TAB. 7.6 – Planification STRIPS classique : Qualité et nombre de problèmes résolus par les planificateurs non optimaux YAHSP, LPG, LAMA et DAEX+YAHSP sur les domaines testés. Dans la colonne `Domain`(x), x est le nombre total d'instances d'un domaine. Les colonnes de la première table indiquent le nombre d'instances résolues (avec en plus pour DAEX+YAHSP le nombre moyen de solutions trouvées sur 11 exécutions). Le nombre moyen des solutions trouvées par LPG ne sont pas indiqués car tous ses exécutions mènent à une solution. Les colonnes de la seconde table représentent les valeurs de la mesure qualitative (et entre parenthèses, pour les planificateurs stochastiques DAEX+YAHSP et LPG, l'*average quality* - cf. section 7.1.1).

Domain	Coverage				
	YAHSP	LPG	LAMA	DAEX+YAHSP	
				1 RUN	6 RUNS
Airport-ipc4 (50)	20	46	37	44 / 9.8	43 / 10
Psr small-ipc4 (50)	50	9	50	50 / 11	50 / 11
Satellite-ipc4 (36)	28	36	32	27 / 11	27 / 11
Openstacks-ipc5 (30)	30	23	30	30 / 10.8	30 / 10.8
Rovers-ipc3 (20)	20	20	20	20 / 11	20 / 11
Zeno-ipc3 (20)	20	20	20	20 / 11	20 / 11
Freecell-ipc3 (20)	20	20	20	20 / 8.5	15 / 10.3
Pathways-ipc5 (30)	30	30	29	30 / 11	30 / 11
Total problems (256)	*218*	*204*	*238*	***241***	*235*

Domain	Quality				
	YAHSP	LPG	LAMA	DAEX+YAHSP	
				1 RUN	6 RUNS
Airport-ipc4 (50)	19.47	4 2.37 (41.1)	35.58	40.34 (38.9)	39.38 (37.9)
Psr small-ipc4 (50)	47.65	9.00 (9.0)	50.00	49.96 (49.9)	49.96 (49.9)
Satellite-ipc4 (36)	16.42	35.98 (35.9)	30.25	26.57 (26.5)	26.57 (26.5)
Openstacks-ipc5 (30)	27.98	22.43 (22.3)	28.55	29.97 (29.8)	29.97 (29.8)
Rovers-ipc3 (20)	17.74	19.93 (19.9)	19.33	19.80 (19.7)	19.80 (19.7)
Zeno-ipc3 (20)	15.37	19.45 (19.6)	19.25	18.91 (18.5)	18.91 (18.5)
Freecell-ipc3 (20)	12.50	18.01 (17.9)	19.52	15.68 (14.0)	12.53 (11.4)
Pathways-ipc5 (30)	25.57	29.37 (29.0)	26.78	29.47 (20.4)	29.47 (20.4)
Total quality (256)	*182.72*	*196.56*	*229.26*	***230.70***	*226.59*

TAB. 7.7 – Planification avec coûts : Qualité et nombre de problèmes résolus par les planificateurs non optimaux YAHSP, LAMA et DAEX+YAHSP sur les domaines testés. Dans la colonne `Domain`(x), x est le nombre total d'instances d'un domaine. Les colonnes de la première table indiquent le nombre d'instances résolues (avec en plus DAEX+YAHSP le nombre moyen de solutions trouvées sur 11 exécutions). Les colonnes de la seconde table représentent les valeurs de la mesure qualitative (et entre parenthèses, pour le planificateur stochastique DAEX+YAHSP, l'*average quality* - cf. section 7.1.1).

Domain	Coverage			
	YAHSP	LAMA	DAEX+YAHSP	
			1 RUN	6 RUNS
Woodworking (30)	20	30	27 / 8.9	21 / 10.8
Pegsolitaire (30)	30	30	30 / 10.9	30 / 10.9
Parcprinter (30)	28	22	28 / 11	28 / 11
Openstacks (30)	30	30	30 / 11	30 /11
Transport (30)	30	30	30 / 11	30 / 11
Scanalyser (30)	27	30	27 / 11	27 / 11
Elevator (30)	30	24	30 / 11	30 / 11
Sokoban (30)	24	25	20 / 9.6	18 / 10.3
Total problems (240)	*219*	*221*	***222***	*214*

Domain	Quality			
	YAHSP	LAMA	DAEX+YAHSP	
			1 RUN	6 RUNS
Woodworking (30)	15.96	24.36	24.79 (24.3)	20.35 (19.6)
Pegsolitaire (30)	20.90	26.19	28.11 (27.2)	28.11 (27.2)
Parcprinter (30)	16.87	11.94	27.25 (17.0)	27.25 (17.0)
Openstacks (30)	8.52	20.73	19.45 (18.2)	19.45 (18.2)
Transport (30)	16.73	26.40	24.99 (23.0)	24.99 (23.0)
Scanalyser (30)	13.68	25.88	21.85 (20.9)	21.85 (20.9)
Elevator (30)	9.60	22.65	18.31 (16.3)	18.31 (16.3)
Sokoban (30)	21.32	24.25	19.79 (19.3)	17.79 (17.4)
Total quality (240)	*123.58*	*182.41*	***184.55***	*178.1*

TAB. 7.8 – Planification temporelle : Qualité et nombre de problèmes résolus par les planificateurs non optimaux YAHSP, LPG, TFD et DAEX+YAHSP sur les domaines testés. Dans la colonne `Domain`(x), x est le nombre total d'instances d'un domaine. Les colonnes de la première table indiquent le nombre d'instances résolues (avec en plus DAEX+YAHSP le nombre moyen de solutions trouvées sur 11 exécutions). Le nombre moyen des solutions trouvées par LPG n'est pas indiqué car tous ses exécutions mènent à une solution. Les colonnes de la seconde table représentent les valeurs de la mesure qualitative (et entre parenthèses, pour les planificateurs stochastiques DAEX+YAHSP et LPG, l'*average quality* - cf. section 7.1.1).

Domain	Coverage				
	YAHSP	LPG	TFD	DAEX+YAHSP	
				1 RUN	6 RUNS
Crewplanning-ipc6 (30)	30	12	29	30 / 11	30 / 11
Elevator-ipc6 (30)	30	30	17	30 / 11	30 / 11
Openstacks-ipc6 (30)	30	30	30	30 / 11	30 / 11
Pegsolitaire-ipc6 (30)	30	30	28	30 / 11	30 / 11
Parcprinter-ipc6 (30)	15	20	15	22 / 10.1	20 / 10.8
Sokoban-ipc6 (30)	22	16	17	17 / 10.5	16 / 11
Rovers-ipc3 (20)	20	20	6	20 / 11	20 / 11
Satellite-ipc3 (20)	20	20	20	20 / 11	20 / 11
Zeno-ipc3 (20)	20	20	20	20 / 11	20 / 11
Total problems (240)	*217*	*198*	*182*	***219***	*216*

Domain	Quality				
	YAHSP	LPG	TFD	DAEX+YAHSP	
				1 RUN	6 RUNS
Crewplanning-ipc6 (30)	24.55	12.00 (12.0)	28.76	29.90 (29.5)	29.90 (29.5)
Elevator-ipc6 (30)	8.31	25.83 (24.6)	13.45	23.24 (20.2)	23.24 (20.2)
Openstacks-ipc6 (30)	17.90	29.45 (27.6)	26.49	28.41 (27.8)	28.41 (27.8)
Pegsolitaire-ipc6 (30)	27.25	29.74 (28.4)	26.78	30.00 (29.8)	30.00 (29.8)
Parcprinter-ipc6 (30)	10.98	19.36 (19.2)	10.27	14.60 (14.2)	13.87 (13.6)
Sokoban-ipc6 (30)	17.20	11.14 (11.1)	12.74	15.60 (15.3)	14.68 (14.6)
Rovers-ipc3 (20)	17.74	19.95 (19.8)	5.78	19.86 (19.8)	19.86 (19.8)
Satellite-ipc3 (20)	6.33	20.00 (19.8)	12.55	16.86 (16.2)	16.86 (16.2)
Zeno-ipc3 (20)	9.70	18.98 (18.4)	11.62	17.50 (16.7)	17.50 (16.7)
Total quality (240)	*139.96*	*186.46*	*148.44*	***195.97***	*194.32*

TAB. 7.9 – Planification STRIPS classique : ratios $\frac{Quality\ Score}{Coverage}$ des tables 7.6 pour chaque domaine ainsi que la moyenne de ces ratios pour tous les types de problèmes.

Domain	Quality/ Total of solved problems				
	YAHSP	LPG	LAMA	DAEX+YAHSP	
				1 RUN	6 RUNS
Airport-ipc4 (100%)	97.35%	92.10%	96.16%	91.69%	91.58%
Psr small-ipc4 (100%)	95.30%	100%	100%	99.91%	99.91%
Satellite-ipc4 (100%)	58.66%	99.95%	94.54%	98.40%	98.40%
Openstacks-ipc5 (100%)	93.28%	97.52%	95.16%	99.89%	99.89%
Rovers-ipc3 (100%)	88.71%	99.65%	96.63%	99.02%	99.02%
Zeno-ipc3 (100%)	76.86%	97.27%	96.23%	94.54%	94.54%
Freecell-ipc3 (100%)	62.52%	90.05%	97.62%	78.39%	83.53%
Pathways-ipc5 (100%)	85.25%	97.91%	92.34%	98.25%	98.25%
Mean ratios (100%)	*82.24%*	***96.81%***	*96.09%*	*95.01%*	*95.64%*

TAB. 7.10 – Planification avec coûts : ratios $\frac{Quality\ Score}{Coverage}$ des tables 7.7 pour chaque domaine ainsi que la moyenne de ces ratios pour tous les types de problèmes.

Domain	Quality/ Total of solved problems			
	YAHSP	LAMA	DAEX+YAHSP	
			1 RUN	6 RUNS
Woodworking (100%)	79.82%	81.21%	91.81%	96.90%
Pegsolitaire (100%)	69.66%	87.31%	93.71%	93.71%
Parcprinter (100%)	60.24%	54.27%	97.33%	97.33%
Openstacks (100%)	28.39%	69.12%	64.85%	64.85%
Transport (100%)	55.77%	88.00%	83.30%	83.30%
Scanalyser (100%)	50.66%	86.27%	80.92%	80.92%
Elevator (100%)	32.00%	94.36%	61.05%	61.05%
Sokoban (100%)	88.85%	97.02%	98.96%	98.83%
Mean ratios (100%)	*58.17%*	*82.19%*	***83.99%***	*84.61%*

TAB. 7.11 – Planification temporelle : ratios $\frac{Quality\ Score}{Coverage}$ des tables 7.8 pour chaque domaine ainsi que la moyenne de ces ratios pour tous les types de problèmes.

Domain	Quality/ Total of solved problems				
	YAHSP	LPG	TFD	DAEX+YAHSP	
				1 RUN	6 RUNS
Crewplanning-ipc6 (100%)	81.82%	100%	99.17%	99.68%	99.68%
Elevator-ipc6 (100%)	27.70%	86.12%	79.11%	77.46%	77.46%
Openstacks-ipc6 (100%)	59.66%	98.15%	88.30%	94.71%	94.71%
Pegsolitaire-ipc6 (100%)	90.83%	99.14%	95.63%	100%	100%
Parcprinter-ipc6 (100%)	73.23%	96.82%	68.49%	66.35%	69.35%
Sokoban-ipc6 (100%)	78.20%	69.63%	74.92%	91.78%	91.75%
Rovers-ipc3 (100%)	88.69%	99.75%	96.39%	99.32%	99.32%
Satellite-ipc3 (100%)	31.64%	100%	62.77%	84.28%	84.28%
Zeno-ipc3 (100%)	48.49%	94.92%	58.09%	87.50%	87.50%
Mean ratios (100%)	*64.47%*	***93.84%***	*80.32%*	*89.01%*	*89.34%*

7.6.2 Discussions

D'après ces résultats, pour au moins un succès, à nouveau DAEX+YAHSP résout globalement plus d'instances (92.53% des 736 instances) que YAHSP seul (88.86%), bien plus (91.25%) que LPG-td (82.50%) et TFD (75.83%) pour les problèmes de planification temporelle, beaucoup plus (94.14%) que LPG-td (79.69%) pour les problèmes de planification STRIPS classique, et un peu plus que LAMA pour les problèmes de planification avec coûts. Par conséquent, DAEX+YAHSP a la meilleure mesure de qualité (cf. tables 7.6 7.7 et 7.8 - tableau 2) sur tous les types de problèmes de planification. De plus, DAEX+YAHSP trouve le plus souvent la solution de référence, ou des solutions proches d'au moins 90% de la solution de référence. Remarquons que la différence de qualité entre YAHSP et DAEX+YAHSP est la plus grande sur les problèmes de planification avec coûts. Pour au moins 6 succès, DAEX+YAHSP résout légèrement moins de problèmes (90.35%) mais beaucoup plus que YAHSP seul (88.86%) et bien plus (90%) que LPG-td (82.50%) et TFD (75.83%) pour les problèmes de planification temporelle mais légèrement moins que LAMA et LPG-td dans les autres domaines. De plus, DAEX+YAHSP garde la meilleure mesure de qualité sur les problèmes de planification temporelle (cf. table 7.8 - tableau 2). Toutefois, LAMA obtient la meilleure mesure de qualité sur les autres types de problèmes de planification. Néanmoins, la mesure de qualité obtenue par DAEX+YAHSP est proche à 97.64% (respectivement 98.84%) de LAMA pour les problèmes de planification avec couts (respectivement planification STRIPS classique).

Cependant, bien que DAEX+YAHSP ait obtenu la meilleure mesure de qualité

sur tous les types de problèmes (cf. dernières lignes de tables 7.6 7.7 et 7.8 - tableau 2), LPG-td a le meilleur ratio sur les problèmes de planification STRIPS classique ainsi que les problèmes de planification temporelle.

Enfin, il n'existe pas un meilleur planificateur : même si le planificateur DaEx+YAHSP (respectivement LPG-td) obtient le meilleur ratio pour un type de problème, il existe au moins un autre planificateur parmi ceux testés ici qui obtient de meilleures solutions sur les instances qu'il arrive à résoudre (cf. tables 7.9 7.10 et 7.11). Voir par exemple le domaine `pegsolitaire` (le planificateur DaEx+YAHSP) pour les problèmes de planification temporelle, et les domaines `elevator` (le planificateur LAMA) et `pathways` (le planificateur DaEx+YAHSP) pour les autres types de problèmes.

7.7 Conclusion

Dans ce chapitre, nous avons présenté les résultats obtenus avec les différentes approches de construction des séries d'états. Il est apparu clairement que l'approche orientée de construction des séries d'états est meilleure que l'approche aveugle. De plus, par rapport aux meilleurs planificateurs de la compétition internationale de planification, le planificateur générique Divide-and-Evolve est robuste et a un très bon comportement sur tous les types de problèmes, même avec l'utilisation d'une seule configuration de paramètres. Par conséquent, le planificateur générique Divide-and-Evolve devient ainsi le tout premier planificateur capable de produire des solutions meilleures ou égales aux solutions des meilleurs planificateurs de l'état de l'art sur **tous les types de problèmes** de planification. Il permet aussi l'optimisation des solutions d'un planificateur non optimal en produisant des solutions égales ou très proche (plus de 90%) des valeurs de références (qui peuvent être optimales).

De plus, d'après le chapitre , le racing par instance permet d'améliorer ces résultats. Il faudrait bien sûr donner le même temps aux autres planificateurs (mais en général ils ont atteint leur meilleure performance en moins de 30 minutes).

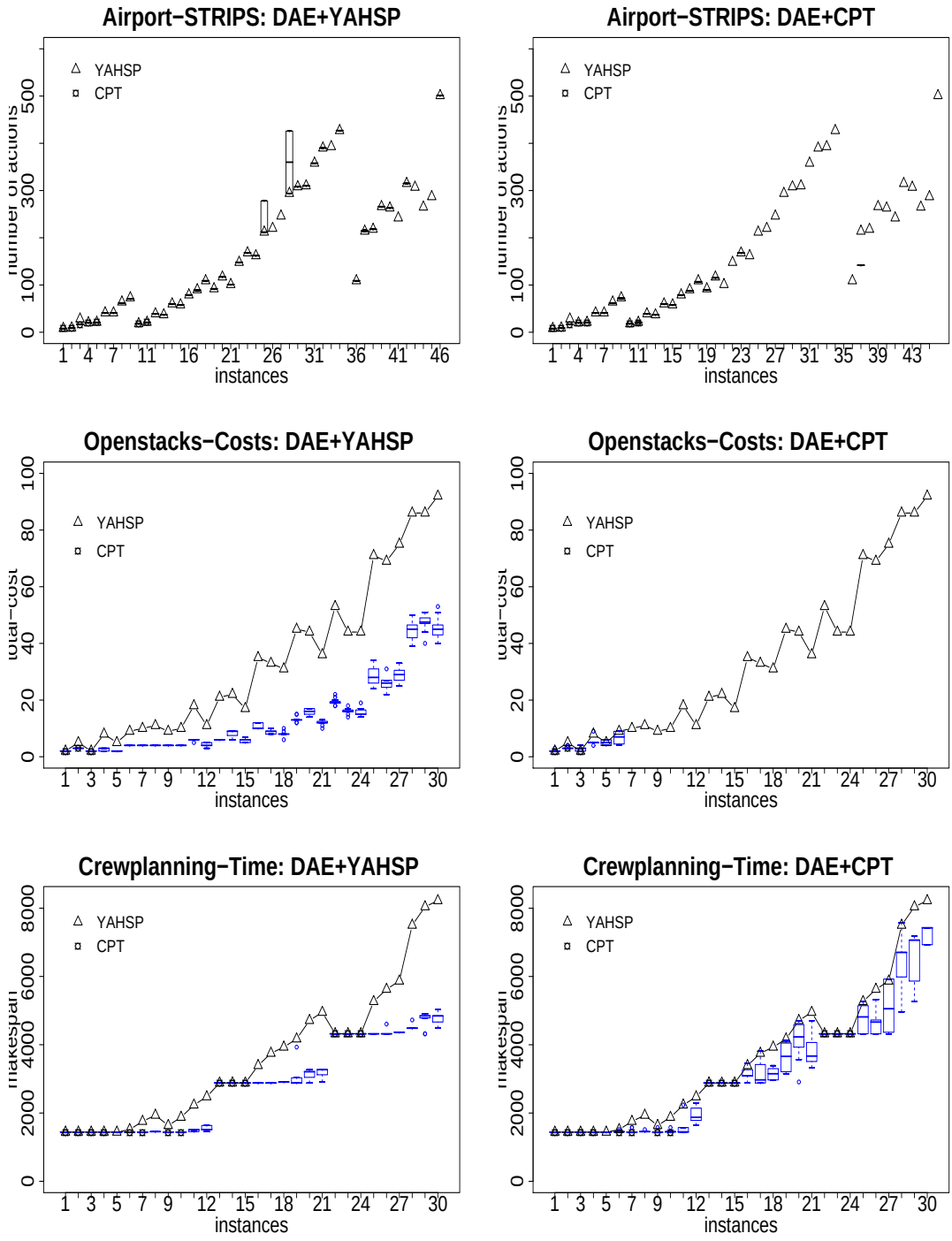

FIG. 7.6 – Qualité optimale de CPT (¤), valeurs de YAHSP (△) et de chaque variante de Divide-and-Evolve sur les domaines `aiport-STRIPS`, `openstacks-Costs` et `crewplanning-Time`.

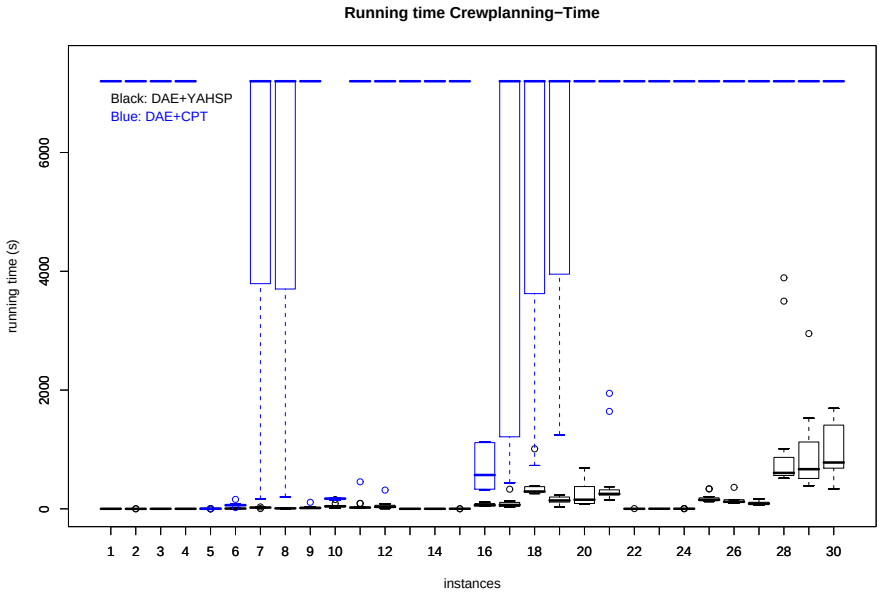

FIG. 7.7 – Distribution des durées d'exécution de DAEX+CPT et DAEX+YAHSP sur les instances du domaine `crewplanning-Time`. La durée maximale d'une exécution de Divide-and-Evolve a été fixée ici à 2 heures.

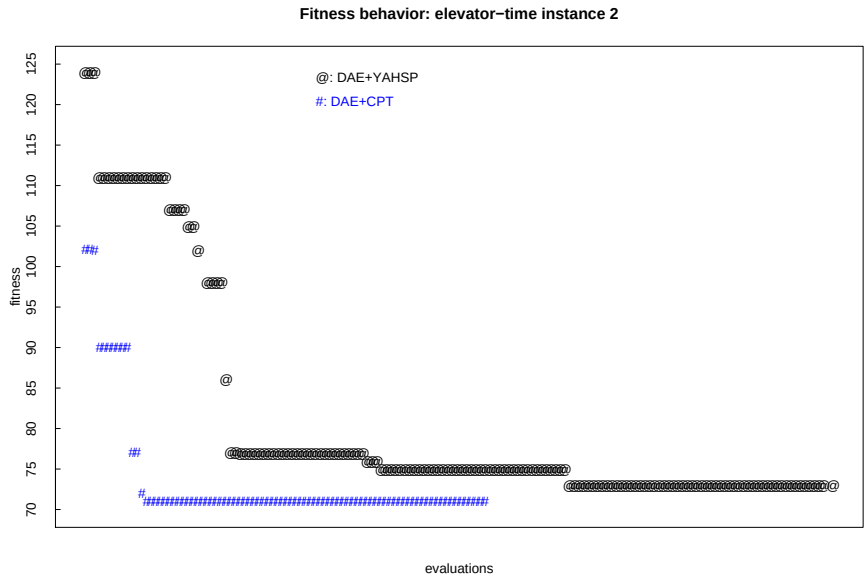

FIG. 7.8 – Exemple de comportement de la fonction d'évaluation de DAEX+CPT (#) et DAEX+YAHSP (@) pour l'instance numéro 2 du domaine `elevator-Time`.

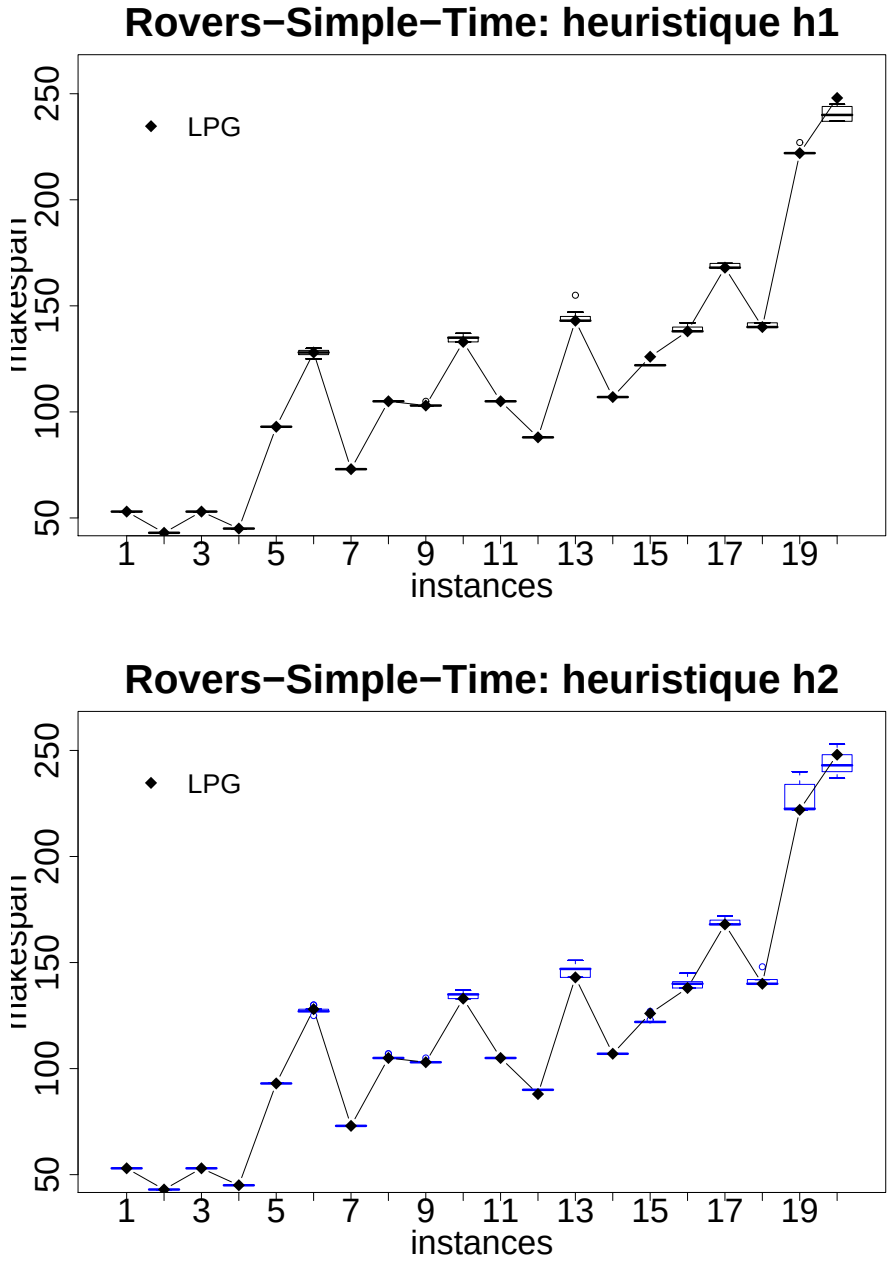

FIG. 7.9 – Impact du choix des heuristiques sur la qualité des solutions de Divide-and-Evolve pour le domaine `rovers simple time`. Divide-and-Evolve+Heuristique h^1 en noir et Divide-and-Evolve+heuristique h^2 en bleu

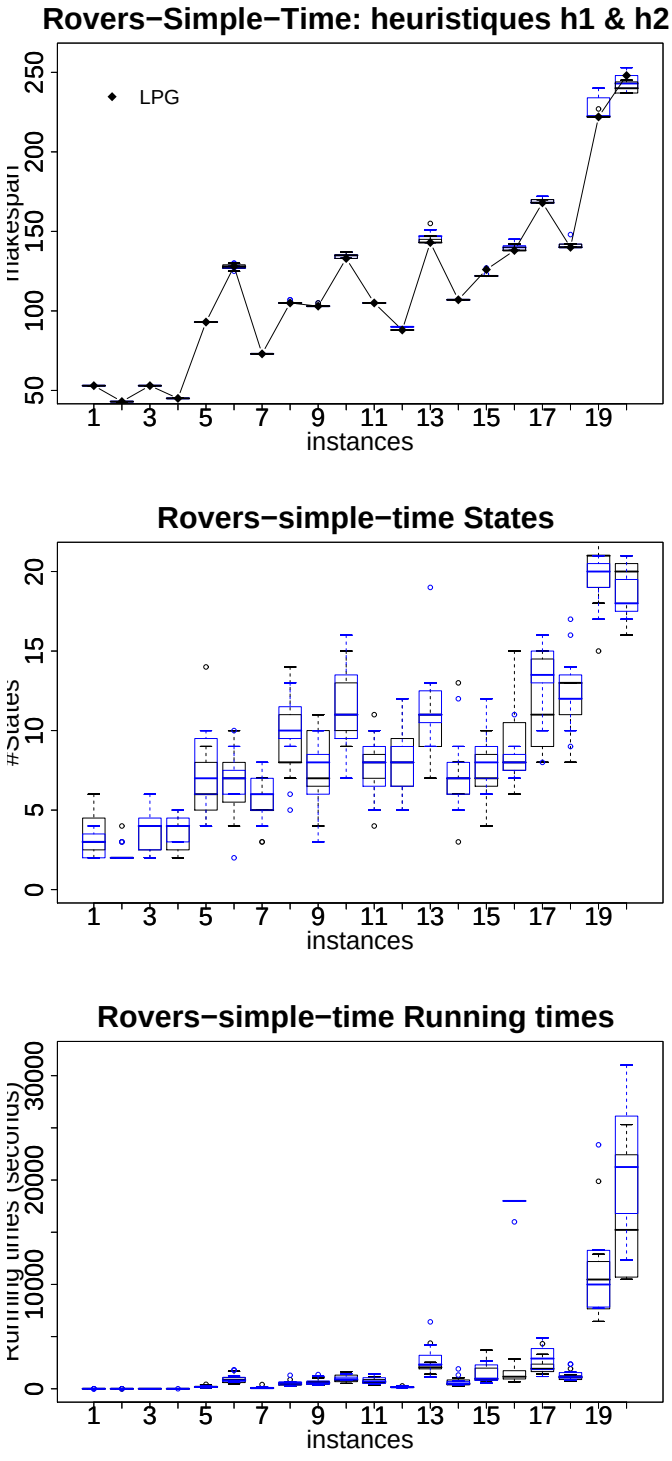

FIG. 7.10 – Superposition des résultats et données statistiques de Divide-and-Evolve avec deux différentes heuristiques sur le domaine `rovers simple time`. Heuristique h^1 en noir et heuristique h^2 en bleu

Chapitre 8

Conclusion

Sommaire

Divide-and-Evolve est un schéma original de collaboration entre les algorithmes évolutionnaires et les planificateurs. Il a été récemment introduit par [Schoenauer 2006, Schoenauer 2007] dans le cadre de la planification temporelle afin de proposer une nouvelle voie pour la résolution des problèmes de planification. Le principe de l'approche Divide-and-Evolve est la génération et l'optimisation des décompositions en séquences d'états intermédiaires d'un problème de planification, à l'aide d'un algorithme évolutionnaire. Une première ébauche de ce schéma, qui préconisait une approche aveugle de construction des séquences d'états, a été établie [Schoenauer 2006, Schoenauer 2007]. Cependant, l'applicabilité et l'efficacité de cette approche n'ont été démontrées que sur trois instances du domaine Zeno[1] (compétition de planification 2000 - voir annexe A) avec l'utilisation de paramètres définis manuellement.

8.1 Contributions

Notre première contribution a donc été sa mise en oeuvre pratique [Bibai 2008b]. Il en est résulté l'implantation du système DAE-1 (cf. chapitre 5 section 5.3) qui a participé à la dernière compétition internationale de planification IPC-2008, devenant ainsi le tout premier planificateur évolutionnaire à y participer depuis sa création. Cependant, la construction aveugle des séquences d'états a soulevé des problèmes de dimensionnement des états et des séquences d'états, ainsi que du choix des atomes (ou des prédicats) pour la construction des états. En effet, le choix d'un des prédicats *in, at, ou fuel-level*, pour la résolution de l'instance numéro 12 du problème zeno par exemple, conduit à des décompositions optimales avec l'utilisation du planificateur embarqué CPT [Vidal 2004a] alors que l'absence du prédicat *fuel-level* mène à des décompositions non-optimales avec le planificateur YAHSP [Vidal 2004b]. Afin de résoudre le problème du choix des atomes, nous avons proposé dans un second temps un choix statistique des prédicats autorisés (cf. chapitre

[1] http ://planning.cis.strath.ac.uk/competition/domains.html

5 section 5.3.1) dans la construction aveugle. Cependant, les mêmes problèmes de dimensionnement et pertinence des atomes n'étaient toujours pas résolus.

Afin de nous affranchir de ces paramètres critiques de l'approche de construction historique, notre troisième contribution a été l'introduction (cf. chapitre 5 section 5.4) d'une nouvelle méthode de construction des séquences d'états dite de *construction orientée dates*. Cette méthode de construction des séquences d'états permet en outre d'apporter une réponse à l'un des challenges actuels de la planification [Benton 2010] à savoir : comment choisir dans un arbre le noeud à explorer si tous les noeuds ont la même valeur heuristique ?

Après nos expérimentations, il est clairement apparu que l'approche orientée dates de construction des séries d'états est meilleure que l'approche aveugle historique. Elle permet par exemple, de trouver la décomposition optimale de l'instance 12 du problème zeno quelle que soit la nature du planificateur embarqué utilisé. De plus, par rapport aux meilleurs planificateurs actuels de la compétition internationale de planification, le planificateur générique Divide-and-Evolve ainsi construit est robuste et a un très bon comportement sur tous les types de problèmes et celà même avec l'utilisation d'une seule et unique configuration de paramètres. Le système Divide-and-Evolve devient donc ainsi, le tout premier planificateur capable de produire des solutions meilleures ou égales aux solutions des meilleurs planificateurs actuels de l'état de l'art sur tous les types de problèmes de planification. Il permet aussi l'optimisation des solutions d'un planificateur non-optimal en produisant des solutions égales ou très proche (plus de 90%) des valeurs de référence (qui peuvent être optimales). De plus, ces résultats peuvent être grandement améliorés si l'on effectue un choix des paramètres sur chaque instance (cf. chapitre 7.7).

8.2 Perspectives

Bien que nos travaux ont montré l'efficacité de Divide-and-Evolve sur tous les types de problèmes de planification STRIPS, ces résultats ont été obtenus avec une configuration de paramètres résultant du *Racing* [Birattari 2002] réduit à un plan d'expérience minimal. Une amélioration des résultats de ce travail pourrait donc être faite par exemple en optimisant les configurations des paramètres de Divide-and-Evolve à l'aide d'une autre méthode telle que *SPO* [Bartz-Beielstein 2005, Hutter 2009] ou *REVAC* [Nannen 2007]. Cependant, l'application de ces méthodes nécessitera probablement, pour le *Racing* par exemple, la définition d'une autre mesure de performance afin de prendre en compte le très faible écart entre les résultats obtenus avec plusieurs configurations de paramètres différents.

On pourrait aussi envisager d'étendre ces résultats aux problèmes de planification non observables en s'inspirant de la méthode de construction orientée des séquences états pour la mise au point de l'algorithme Divide-and-Evolve et en embarquant un planificateur non-optimal.

Une autre piste d'amélioration serait la caractérisation de la difficulté estimée

d'un problème de planification donné. Nous avons proposé dans ce travail de la définir à l'aide d'un sondage de la population initiale avec l'utilisation d'une très faible contrainte sur le planificateur embarqué. Cependant, si cette contrainte est très faible, ce sondage peut prendre en fonction de la population initiale une durée importante. A contrario, si elle est trop forte, l'algorithme pourrait ne pas trouver une solution au problème car le planificateur embarqué aurait été trop bridé. Un choix judicieux de contrainte augmenterait à coup sûr l'efficacité de Divide-and-Evolve et réduirait en même temps sa durée d'exécution.

Enfin, l'efficacité de l'approche Divide-and-Evolve démontrée empiriquement dans ce travail ouvre un boulevard pour la résolution des problèmes de planifications multi-objectifs. Car l'utilisation du calcul évolutionnaire pour l'optimisation au sens de Pareto est nécessaire pour la résolution des problèmes de planification multi-objectifs, comme en témoignent les résultats préliminaire de [Schoenauer 2006, Schoenauer 2007].

Annexe A

Zeno simple time domain

Dans ce domaine, les avions consomment des unités (*flevel*) de carburant pour se déplacer. Le reservoirs peuvent être remis à niveau en utilisant un opérateur de réappovisionnement en combustible : *refuel*. Il y a un certain nombre de villes (*city*), un certain nombre d'avions (*aircraft*), un certain nombre de personnes (*person*), et un certain nombre de niveaux possibles de carburant (*fuel-level*). L'objectif consiste à transporter un ensemble de personnes d'un endroit à un autre.

```
(define (domain zeno-travel)
(:predicates
 (at ?x ?c) (in ?p ?a) (fuel-level ?a ?l)
         (lnext ?l1 ?l2)(aircraft ?p) (person ?x)
         (city ?x) (flevel ?x) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:action board
 :parameters ( ?p ?a ?c)
 :duration 20
 :precondition
(and (person ?p) (aircraft ?a) (city ?c)  (at ?p ?c) (at ?a ?c))
 :effect
(and (in ?p ?a) (not (at ?p ?c))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:action debark
 :parameters ( ?p ?a ?c)
 :duration 30
 :precondition
(and (person ?p) (aircraft ?a) (city ?c)  (in ?p ?a) (at ?a ?c))
 :effect
(and (at ?p ?c) (not (in ?p ?a))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(:action fly
 :parameters ( ?a ?c1 ?c2 ?l1 ?l2)
 :duration 180
 :precondition
(and (aircraft ?a) (city ?c1) (city ?c2) (flevel ?l1) (flevel ?l2)
        (at ?a ?c1) (fuel-level ?a ?l1) (lnext ?l2 ?l1))
 :effect
(and (at ?a ?c2) (fuel-level ?a ?l2) (not (at ?a ?c1)) (not (fuel-level ?a ?l1))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:action zoom
 :parameters ( ?a ?c1 ?c2 ?l1 ?l2 ?l3)
 :duration 100
 :precondition
(and (aircraft ?a) (city ?c1) (city ?c2) (flevel ?l1) (flevel ?l2) (flevel ?l3)
        (at ?a ?c1) (fuel-level ?a ?l1) (lnext ?l2 ?l1) (lnext ?l3 ?l2))
 :effect
(and (at ?a ?c2) (fuel-level ?a ?l3) (not (at ?a ?c1)) (not (fuel-level ?a ?l1))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:action refuel
 :parameters ( ?a ?c ?l ?l1)
 :duration 73
 :precondition
(and (aircraft?a) (city ?c) (flevel ?l) (flevel ?l1)
        (fuel-level ?a ?l) (lnext ?l ?l1) (at ?a ?c))
 :effect
(and (fuel-level ?a ?l1) (not (fuel-level ?a ?l))))

)
```

Annexe B

Courriel portant sur la version mise à jour de TFD

```
Date: Mon, 02 Nov 2009 14:17:32 +0100
From: =?ISO-8859-1?Q?Robert_Mattm=FCller?=
 <mattmuel@informatik.uni-freiburg.de>
To: jacques bibai <jacques.bibai@thalesgroup.com>
CC: Patrick Eyerich <eyerich@informatik.uni-freiburg.de>,
 =?ISO-8859-1?Q?Gabi_R=F6ger?= <roeger@informatik.uni-freiburg.de>
Subject: Re: Reference values of TFD

Hi Jacques.

jacques bibai schrieb:
> I am building a new version of DAE (meaning Divide-and-Evolve) system=20
> which is starting to produce some good results in some IPC6 temporal=20
> satisficing domains. In order to compare and improve the DAE system, I=20
> would like to have reference values of TFD in all those problems. Pleas=
e=20
> can you send me those values and/or the current version of TFD if it is=
=20
> possible?

First of all I'd like to apologize for not sending you the TFD source=20
code right after we talked about it at ICAPS.

Anyway: You can find the latest TFD sources at=20
http://tfd.informatik.uni-freiburg.de/. The latest release is=20
tfd-src-0.2.1.tgz.

If you encounter any difficulties with the code, just let me know.

Best regards,
Robert.

--=20
Robert Mattm=FCller
```

Department of Computer Science University of Freiburg
Georges-Koehler-Allee, Bldg 52 79110 Freiburg, Germany
Phone: +49 761 203-8229 Fax: +49 761 203-8222
mailto:mattmuel@informatik.uni-freiburg.de
http://www.informatik.uni-freiburg.de/~mattmuel

Annexe C

Courriel portant sur la version mise à jour de LAMA

Subject: Re: Reference values of LAMA
Date: Thu, 8 Oct 2009 11:47:51 +1000
From: Silvia Richter <silvia.richter@nicta.com.au>
To: jacques bibai <jacques.bibai@thalesgroup.com>

Hi Jacques,

you can download the latest version of LAMA here:

http://www.informatik.uni-freiburg.de/~srichter/software/lama.tar.gz

To download that an re-run it is probably the best option. It has not changed much since the competition, only some minor fixes.

You could also access LAMA's results in the competition from the competition web page at
http://ipc.informatik.uni-freiburg.de/
(look for "Raw competition results" under the heading "Post-IPC"), but then you'd not be comparing on the same hardware of course.

Cheers,
Silvia

jacques bibai wrote:
> Hi Silvia,
> I am building a new version of DAE (meaning Divide-and-Evolve) system
> which is starting to produce some good results in some IPC6 sequential
> satisficing domains. In order to compare and improve the DAE system, I
> would like to have reference values of LAMA in all those problems.
> Please can you send me those values and/or the current version of LAMA
> if it is possible?
> Thanks a lot.
>

> Jacques
>
>

Annexe D

Article [Bibai 2009c]

Bibliographie

[Aler 1998] Ricardo Aler, Daniel Borrajo et Pedro Isasi. *Evolving heuristics for planning*. Evolutionary Programming VII, pages 745–754, 1998. 40

[Aler 2001] Ricardo Aler, Daniel Borrajo et Pedro Isasi. *Learning to Solve Planning Problems Efficiently by Means of Genetic Programming*. Evolutionary Computation, vol. 9, no. 4, pages 387–420, 2001. PMID : 11709102. 40

[Amir 2003] Eyal Amir et Barbara Engelhardt. *Factored planning*. In IJCAI'03 : Proceedings of the 18th international joint conference on Artificial intelligence, pages 929–935, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. 41

[Bäckström 1998] Christer Bäckström. *Computational Aspects of Reordering Plans*. Journal of Artificial Intelligence Research, vol. 9, pages 99–137, 1998. 49

[Bartz-Beielstein 2005] Th. Bartz-Beielstein, C.W.G. Lasarczyk et M. Preuss. *Sequential Parameter Optimization*. In Bob McKay, editeur, Proc. CEC'05, pages 773– 780. IEEE Press, 2005. 68, 114

[Benton 2010] J. Benton, Kartik Talamadupula, Patrick Eyerich, Robert Mattmuller et Subbarao Kambhampati. *G-Value Plateaus : A Challenge for Planning*, 2010. 114

[Bibai 2008a] Jacques Bibai, Pierre Savéant, Marc Schoenauer et Vincent Vidal. DAE : Planning as Artificial Evolution (Deterministic part). At International Planning Competition (IPC) `http ://ipc.icaps-conference.org/`, 2008. 53, 57, 58, 87

[Bibai 2008b] Jacques Bibai, Marc Schoenauer, Pierre Savéant et Vidal Vincent. *Planification Evolutionnaire par Décomposition*. Research Report RT-0355, INRIA a CCSD electronic archive server based on P.A.O.L [http ://hal.inria.fr/oai/oai.php] (France), 2008. 57, 113

[Bibai 2009a] Jacques Bibai, Pierre Savéant, Marc Schoenauer et Vincent Vidal. *Learning Divide-and-Evolve Parameter Configurations with Racing*. In Amanda Coles, Andrew Coles, Sergio Jimenez Celorrio, Susana Fernandez Arregui et Tomas de la Rosa, editeurs, ICAPS 2009 proceedings of the Workshop on Planning and Learning, Thessaloniki Grèce, 2009. ICAPS and University of Macedonia, AAAI press. 75

[Bibai 2009b] Jacques Bibai, Marc Schoenauer et Pierre Savéant. *Construction des sèries d'états dans l'algorithme Divide-and-Evolve*. In Dixième conférence de la Société Francaise de Recherche Opèrationnelle et d'Aide à la Dècision , Livre des résumès, pages 98–99, Nancy France, 2009. ROADEF'09. 53, 62

[Bibai 2009c] Jacques Bibai, Marc Schoenauer et Pierre Savéant. *Divide-And-Evolve Facing State-of-the-art Temporal Planners during the* 6^{th} *International Planning Competition*. In C. Cotta et P. Cowling, editeurs, Ninth European

Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2009), numéro 5482 de Lecture Notes in Computer Science, pages 133–144. Springer-Verlag, 2009. vi, 53, 62, 96, 123

[Bibai 2010a] Jacques Bibai, Pierre Savéant, Marc Schoenauer et Vincent Vidal. *An Evolutionary Metaheuristic Based on State Decomposition for Domain-Independent Satisficing Planning.* In ICAPS 2010, pages 18–25. AAAI press, 2010. 45, 53, 62, 87, 100

[Bibai 2010b] Jacques Bibai, Pierre Savéant, Marc Schoenauer et Vincent Vidal. *On the Benefit of Sub-Optimality within the Divide-and-Evolve Scheme.* In P. Cowling et P. Merz, editeurs, EvoCOP 2010, numéro 6022 de Lecture Notes in Computer Science, pages 23–34. Springer-Verlag, 2010. 53, 96

[Bibai 2010c] Jacques Bibai, Pierre Savéant, Marc Schoenauer et Vincent Vidal. *On the Generality of Parameter Tuning in Evolutionary Planning.* In Genetic and Evolutionary Computation Conference (GECCO 2010). ACM, 2010. 75

[Birattari 2002] Mauro Birattari, Thomas Stützle, Luis Paquete et Klaus Varrentrapp. *A Racing Algorithm for Configuring Metaheuristics.* In GECCO '02 : Proceedings of the Genetic and Evolutionary Computation Conference, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. 68, 69, 74, 114

[Blum 1995] Avrim Blum et Merrick Furst. *Fast Planning Through Planning Graph Analysis.* In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95), pages 1636–1642, Août 1995. 8, 14, 20

[Blum 1997] Avrim L. Blum et Merrick L. Furst. *Fast planning through planning graph analysis.* Artificial Intelligence, vol. 90, no. 1–2, pages 279–298, 1997. 8, 14, 17, 18, 20, 24

[Bonet 1997] Blai Bonet, Gábor Loerincs et Hector Geffner. *A Robust and Fast Action Selection Mechanism for Planning.* In AAAI/IAAI, pages 714–719, 1997. 22

[Bonet 1998] Blai Bonet et Héctor Geffner. *HSP : Heuristic Search Planner.* Rapport technique, 1998. 9, 22

[Bonet 1999] Blai Bonet et Hector Geffner. *Planning as Heuristic Search : New Results.* In Susanne Biundo et Maria Fox, editeurs, ECP, volume 1809 of *Lecture Notes in Computer Science*, pages 360–372. Springer, 1999. 9, 22

[Bonet 2001a] Blai Bonet et Hector Geffner. *Heuristic Search Planner 2.0.* AI Magazine, vol. 22, no. 3, pages 77–80, 2001. 9, 22

[Bonet 2001b] Blai Bonet et Héctor Geffner. *Planning as Heuristic Search.* Artificial Intelligence, vol. 129, no. 1–2, pages 5–33, 2001. 22

[Borrajo 1997] Daniel Borrajo et Manuela Veloso. *Lazy Incremental Learning of Control Knowledge for Efficiently Obtaining Quality Plans.* Artificial Intelligence Review, vol. Volume 11, pages 371–405, 1997. 40

[Brafman 2006] Ronen I. Brafman et Carmel Domshlak. *Factored planning : how, when, and when not.* In AAAI'06 : Proceedings of the 21st national conference on Artificial intelligence, pages 809–814. AAAI Press, 2006. 41

[Brié 2005] A. H. Brié et P. Morignot. *Genetic Planning Using Variable Length Chromosomes.* In Proc. ICAPS, 2005. 9, 39, 40

[Bylander 1994a] T. Bylander. *The Computational Complexity of Propositional STRIPS Planning.* Artificial Intelligence, vol. 69, no. 1-2, pages 165–204, 1994. 14, 39

[Bylander 1994b] Tom Bylander. *The Computational Complexity of Propositional STRIPS Planning.* Artificial Intelligence, vol. 69, no. 1–2, pages 165–204, 1994. 21

[Chapman 1987] David Chapman. *Planning for conjunctive goals.* Artificial Intelligence, July 1987, vol. 32, no. 3, pages 281–331, 1987. 8

[Chen 2006] Yixin Chen, Benjamin W. Wah et Chih-Wei Hsu. *Temporal Planning using Subgoal Partitioning and Resolution in SGPlan.* J. Artif. Intell. Res. (JAIR), vol. 26, pages 323–369, 2006. 20, 42

[Cushing 2007] William Cushing, Daniel S. Weld, Subbarao Kambhampati, Mausam et Kartik Talamadupula. *Evaluating Temporal Planning Domains.* In Mark S. Boddy, Maria Fox et Sylvie Thiébaux, editeurs, ICAPS, pages 105–112. AAAI, 2007. 42

[Desquilbet 1992] C. Desquilbet. *Détermination de trajets optimaux par algorithmes génétiques.* Rapport de stage d'option B2 de l'Ecole Polytechnique. Palaiseau, France, Juin 1992. Advisor : Marc Schoenauer. In French. 42

[Dijkstra 1971] Edsger W. Dijkstra. *A short introduction to the art of programming.* 1971. 27

[Eiben 2003] A.E. Eiben et J.E. Smith. Introduction to evolutionary computing. Springer Verlag, 2003. 39

[Eiben 2007] A. E. Eiben, Z. Michalewicz, M. Schoenauer et J. E. Smith. *Parameter Control in Evolutionary Algorithms.* In Lipcoll et al. [Lobo 2007], chapter 2, pages 19–46. 67

[Eyerich 2009] Patrick Eyerich, Robert Mattmüller et Gabriele Röger. *Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning.* In 19^{th} Int. Conf. on Automated Planning and Scheduling (ICAPS 2009), pages 130–137, 2009. 26, 29, 42, 101

[Fikes 1971] R. Fikes et N. Nilsson. *STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence, vol. 1, pages 27–120, 1971. 2, 8

[Fogel 1966] L. J. Fogel, A. J. Owens et M. J. Walsh. Artificial intelligence through simulated evolution. New York : John Wiley, 1966. 31

[Fogel 1995] D.B. Fogel. *Evolutionary Computation. Toward a New philosophy of Machine Intelligence.* IEEE Press, 1995. 31

[Fox 1998] Maria Fox et Derek Long. *The Automatic Inference of State Invariants in TIM.* J. Artif. Intell. Res. (JAIR), vol. 9, pages 367–421, 1998. 19, 20

[Fox 1999] Maria Fox et Derek Long. *The Detection and Exploitation of Symmetry in Planning Problems.* In Dean Thomas, editeur, Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2), pages 956–961, S.F., Juillet 31–Août 6 1999. Morgan Kaufmann Publishers. 9, 19, 20

[Fox 2000] Maria Fox et Derek Long. *Utilizing Automatically Inferred Invariants in Graph Construction and Search.* In AIPS, pages 102–111, 2000. 20

[Fox 2003a] M. Fox et D. Long. *PDDL2.1 : An extension to PDDL for expressing temporal planning domains.* Journal of Artificial Intelligence Research, vol. 20, pages 61–124, 2003. 2

[Fox 2003b] M. Fox et D. Long. *PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains.* Journal of Artificial Intelligence Research, vol. 20, pages 61–124, 2003. 2

[Gerevini 2003a] A. Gerevini, A. Saetti et I. Serina. *On Managing Temporal Information for Handling Durative Actions in LPG.* In AI*IA 2003 : Advances in Artificial Intelligence. Springer Verlag, 2003. 9, 27, 28, 42, 77, 101

[Gerevini 2003b] A. Gerevini, A. Saetti et I. Serina. *Planning through Stochastic Local Search and Temporal Action Graphs in LPG.* Journal of Artificial Intelligence Research, vol. 20, pages 239–290, 2003. 9, 27, 28, 42, 77, 101

[Goldberg 1989] D. E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, 1989. 31

[Grefenstette 1986] J. J. Grefenstette. *Optimization of Control Parameters for Genetic Algorithms.* IEEE Trans. on Systems, Man and Cybernetics, vol. SMC-16, 1986. 68

[Hansen 2007] Eric A. Hansen et Rong Zhou. *Anytime Heuristic Search.* J. Artif. Intell. Res. (JAIR), vol. 28, pages 267–297, 2007. 26

[Hart 1968] P. Hart, N. Nilsson, B. et Raphael. *A Formal Basis for the Heuristic Determination of Minimum-Cost Paths.* IEEE Trans. on Systems Science and Cybernetics, vol. SSC-4, no. 2, pages 100–107, Juillet 1968. 21

[Haslum 2000] P. Haslum et H. Geffner. *Admissible Heuristics for Optimal Planning.* In Proc. AIPS-2000, pages 70–82, 2000. 23, 24, 62, 88, 100

[Helmert 2006] Malte Helmert. *The Fast Downward Planning System.* J. Artif. Intell. Res. (JAIR), vol. 26, pages 191–246, 2006. 9, 21, 26

[Helmert 2008] Malte Helmert. Understanding planning tasks. Springer Verlag, 2008. 14, 101

[Hillier 2001] F. Hillier et G. Lieberman. Introduction to operations research. McGraw-Hill, Boston. 7th edition, 2001. 28

[Hoffmann 2001] J. Hoffmann et B. Nebel. *The FF Planning System : Fast Plan Generation Through Heuristic Search*. Journal of Artificial Intelligence Research, vol. 14, pages 253–302, 2001. 9, 21, 22, 25, 26

[Hoffmann 2004] Jörg Hoffmann, Julie Porteous et Laura Sebastia. *Ordered Landmarks in Planning*. Journal of Artificial Intelligence Research, vol. 22, pages 215–278, 2004. 24, 41

[Holland 1975] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. 31

[Hutter 2009] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown et Kevin P. Murphy. *An experimental investigation of model-based parameter optimisation : SPO and beyond*. In Franz Rothlauf et al., editeur, GECCO'09, pages 271–278. ACM, 2009. 68, 114

[Karpas 2009] Erez Karpas et Carmel Domshlak. *Cost-Optimal Planning with Landmarks*. In Craig Boutilier, editeur, IJCAI, pages 1728–1733, 2009. 24, 42

[Kautz 1999] Henry Kautz et Bart Selman. *Unifying SAT-Based and Graph-Based Planning*. In Thomas Dean, editeur, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pages 318–325, San Francisco, 1999. Morgan Kaufmann. 20

[Kelareva 2007] E. Kelareva, O. Buffet, J. Huang et S. Thiébaux. *Factored Planning using Decomposition Trees*. In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07), January 2007. 41

[Keyder 2008] Emil Keyder et Hector Geffner. *Heuristics for Planning with Action Costs Revisited*. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis et Nikolaos M. Avouris, editeurs, ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 588–592. IOS Press, 2008. 22, 26

[Knoblock 1994] Craig A. Knoblock. *Automatically Generating Abstractions for Planning*. Artificial Intelligence, vol. 68, no. 2, pages 243–302, 1994. 41

[Koehler 1997] Jana Koehler, Bernhard Nebel, Jörg Hoffmann et Yannis Dimopoulos. *Extending Planning Graphs to an ADL Subset*. In Sam Steel et Rachid Alami, editeurs, ECP, volume 1348 of *Lecture Notes in Computer Science*, pages 273–285. Springer, 1997. 9

[Koehler 1999] Jana Koehler et Joerg Hoffmann. *Handling of Inertia in a Planning System*. report00122, Institut für Informatik, Universität Freiburg, Mai 14 1999. 19

[Koehler 2000] Jana Koehler et Jörg Hoffmann. *On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm*. Journal of Artificial Intelligence Research, vol. 12, pages 338–386, 2000. 24, 41

[Koffka 1935] K. Koffka. Principles of gestalt psychology. Book, 1935. 1

[Koza 1992] John R. Koza. Genetic Programming : on the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA, 1992. 9, 32, 39

[Koza 1994] J. R. Koza. *Genetic Programing II : Automatic Discovery of Reusable Programs*. 1994. 32

[Lansky 1995] Amy L. Lansky et Lise Getoor. *Scope and Abstraction : Two Criteria for Localized Planning*. In IJCAI, pages 1612–1619, 1995. 41

[Levine 1999] John Levine et David Humphreys. *Learning Action Strategies for Planning Domains Using Genetic Programming*. In APPLICATIONS OF EVOLUTIONARY COMPUTING, EVOWORKSHOPS2003, pages 684–695. Kluwer Academic Publishers, 1999. 40

[Levine 2009] John Levine, Henrik Westerberg, Michelle Galea et Dave Humphreys. *Evolutionary-based learning of generalised policies for AI planning domains*. In GECCO '09 : Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pages 1195–1202, New York, NY, USA, 2009. ACM. 40

[Lobo 2007] F.G. Lobo, C.F. Lima et Z. Michalewicz, editeurs. Parameter setting in evolutionary algorithms. Springer, Berlin, 2007. 67, 127

[Maron 1994] Oded Maron et Andrew W. Moore. *Hoeffding Races : Accelerating Model Selection Search for Classification and Function Approximation*. In Advances in neural information processing systems 6, pages 59–66. Morgan Kaufmann, 1994. 69

[McAllester 1991] D. McAllester et D. Rosenblitt. *Systematic Nonlinear Planning*. In Proc. AAAI-91, pages 634–639, 1991. 27

[McDermott 1998] D. McDermott. PDDL – The Planning Domain Definition language. At `http ://ftp.cs.yale.edu/pub/mcdermott`, 1998. 2

[Moscato 2003] Pablo Moscato et Carlos Cotta. *A Gentle Introduction to Memetic Algorithms*. In Bernhard Nebel, editeur, Handbook of Metaheuristics, pages 105–144. springer, 2003. 40

[Muslea 1997] Ion Muslea. *SINERGY : A Linear Planner Based on Genetic Programming*. In ECP '97 : Proceedings of the 4^{th} European Conference on Planning, pages 312–324, London, UK, 1997. Springer-Verlag. 9, 39, 40

[Nannen 2007] V. Nannen et A. E. Eiben. *Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters*. In M. Veloso, editeur, Proc. Intl. Joint Conference on Artificial Intelligence, pages 975–980, 2007. 68, 114

[Newell 1972] A. Newell et H. Simon. Human problem solving. Prentice-Hall, Englewood Cliffs, NJ, 1972. 2

[Newton 2007] Muhammad Abdul Hakim Newton, John Levine, Maria Fox et Derek Long. *Learning Macro-Actions for Arbitrary Planners and Domains*. In ICAPS, pages 256–263, 2007. 40

[Nguyen 2001] XuanLong Nguyen et Subbarao Kambhampati. *Reviving Partial Order Planning.* In Bernhard Nebel, editeur, Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01), pages 459–466, San Francisco, CA, Août 4–10 2001. Morgan Kaufmann Publishers, Inc. 8

[Nguyen 2002] XuanLong Nguyen, Subbaro Kambhampati et Romeo S. Nigenda. *Planning Graph as the Basis for Deriving Heuristics for Plan Synthesis by State Space and CSP Search.* Artificial Intelligence, vol. 135, no. 1–2, pages 73–123, 2002. 9

[Pednault 1989] Edwin P. D. Pednault. *ADL : Exploring the Middle Ground between STRIPS and the Situation Calculus.* In N. S. Sridharan, editeur, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pages 324–332, San Mateo, California, 1989. Morgan Kaufmann. 2, 19

[Penberthy 1992] J. Penberthy et D. Weld. *UCPOP : A sound, complete, partial order planner for ADL.* In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, pages 103–114. Morgan Kaufmann, 1992. 8

[Pohl 1970] Ira Pohl. *Heuristic Search Viewed as Path Finding in a Graph.* Artif. Intell, vol. 1, no. 3, pages 193–204, 1970. 26

[Porteous 2001] Julie Porteous, Laura Sebastia et Joerg Hoffmann. *On the extraction, ordering, and usage of landmarks in planning.* In Proc. EUROPEAN CONF. ON PLANNING, pages 37–48, 2001. 24, 42

[Rechenberg 1973] I. Rechenberg. Evolutionsstrategie. Friedrich Frommann Verlag (Günther Holzboog KG), Stuttgart, 1973. 31

[Regnier 1991] P. Regnier et B. Fade. *Complete determination of parallel actions and temporal optimization in linear plans of actions.* In Proceedings of the European Workshop on Planning (EWSP-91), pages 100–111, Swiss, 1991. Sankt Augustin. 49

[Richter 2008] Silvia Richter, Malte Helmert et Matthias Westphal. *Landmarks Revisited.* In AAAI'08 : Proceedings of the 23^{rd} National Conference on Artificial intelligence, pages 975–982. AAAI Press, 2008. 9, 21, 26, 28, 42, 77, 101

[Rintanen 2007] Jussi Rintanen. *Complexity of Concurrent Temporal Planning.* In ICAPS, pages 280–287, 2007. 14, 39

[Sacerdoti. 1974] E. D. Sacerdoti. *Planning in a hierarchy of abstraction spaces.* Artificial Intelligence, vol. 5, pages 115–135, 1974. 41

[Sacerdoti 1975] E. D. Sacerdoti. *The Nonlinear Nature of Plans.* In Proceedings of the International Joint Conference on Artificial Intelligence, pages 206–214, 1975. 8

[Sacerdoti 1977] E. D. Sacerdoti. A structure for plans and behavior. Elsevier North-Holland, New York, 1977. 8

[Schoenauer 2006] Marc Schoenauer, Pierre Savéant et Vincent Vidal. *Divide-and-Evolve : a New Memetic Scheme for Domain-Independent Temporal Planning*. In J. Gottlieb et G. Raidl, editeurs, Proc. EvoCOP'06. Springer Verlag, 2006. 9, 40, 53, 57, 60, 113, 115

[Schoenauer 2007] Marc Schoenauer, Pierre Savéant et Vincent Vidal. *Divide-and-Evolve : a Sequential Hybridization Strategy using Evolutionary Algorithms*. In Z. Michalewicz et P. Siarry, editeurs, Advances in Metaheuristics for Hard Optimization, pages 179–198. Springer, 2007. 9, 40, 53, 57, 113, 115

[Schoenauer 2010] Marc Schoenauer. *Introduction à l'évolution artificielle*. In Cours - Ecole EA 2010 - Calais, 2010. 40

[Schwefel 1977] H.-P. Schwefel. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. Birkhaeuser, 1977. 31

[Sebastia 2006] L. Sebastia, E. Onaindia et E. Marza. *Decomposition of Planning Problems*. AI Communications, vol. 19(1), pages 49–81, 2006. 41

[Smit 2009] Selmar K. Smit et A. E. Eiben. *Comparing parameter tuning methods for evolutionary algorithms*. In IEEE Congress on Evolutionary Computation, pages 399–406. IEEE, 2009. 70

[Smith 1999] David E. Smith et Daniel S. Weld. *Temporal Planning with Mutual Exclusion Reasoning*. In Dean Thomas, editeur, Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol1), pages 326–337, S.F., Juillet 31–Août 6 1999. Morgan Kaufmann Publishers. 19

[Spector 1994] L. Spector. *Genetic Programming and AI Planning Systems*. In Proc. AAAI 94, pages 1329–1334. AAAI/MIT Press, 1994. 9, 39, 40

[Sussman 1975] G. J. Sussman. A computer model of skill acquisition. American Elsevier, New York, 1975. Volume 1 of Artificial Intelligence Series. 8

[Tate 1975] Austin Tate. *Interacting goals and their use*. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75), pages 215–218, Tbilisi, Georgia, 1975. IJCAII. 8

[Tate 1977] A. Tate. *Generating Project Networks*. In Raj Reddy, editeur, Proceedings of the 5th International Joint Conference on Artificial Intelligence, pages 888–893, Cambridge, MA, 1977. William Kaufmann. 8

[Veloso 1995] Manuela Veloso, Jaime Carbonell, Alicia Perez, Daniel Borrajo, Eugene Fink et Jim Blythe. *Integrating Planning and Learning : The PRODIGY Architecture*. Journal of Experimental and Theoretical Artificial Intelligence, vol. 7, pages 81–120, 1995. 40

[Vidal 2004a] V. Vidal et H. Geffner. *Branching and Pruning : An Optimal Temporal POCL Planner based on Constraint Programming*. In Proc. AAAI, pages 570–577, 2004. 8, 9, 27, 28, 46, 85, 89, 96, 113

[Vidal 2004b] Vincent Vidal. *A Lookahead Strategy for Heuristic Search Planning*. In 14^{th} International Conference on Automated Planning & Scheduling - ICAPS, pages 150–160, 2004. 9, 25, 26, 46, 76, 85, 89, 96, 101, 113

[Vidal 2006] V. Vidal et H. Geffner. *Branching and Pruning : An Optimal Temporal POCL Planner based on Constraint Programming.* Artificial Intelligence, vol. 170, no. 3, pages 298–335, 2006. 46, 77

[Wah 2006] Benjamin W. Wah et Yixin Chen. *Constraint partitioning in penalty formulations for solving temporal planning problems.* Artif. Intell, vol. 170, no. 3, pages 187–231, 2006. 20, 42

[Waldinger 1975] Richard Waldinger. Achieving several goals simultaneously. Ellis Horwood, Chichester, England, 1975. 8

[Warren 1974] D. Warren. *Warplan : a System for Generating Plans.* Memo 76, Computational Logic Dept., School of Artificial Intelligence, University of Edinburgh, 1974. 8

[Weld 1998] Daniel S. Weld, Corin R. Anderson et David E. Smith. *Extending Graphplan to Handle Uncertainty and Sensing Actions.* In Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98), pages 897–904, Menlo Park, Juillet 26–30 1998. AAAI Press. 9

[Weld 1999] Daniel S. Weld. *Recent Advances in AI Planning.* AI Magazine, vol. 20, no. 2, pages 93–123, 1999. 18

[Westerberg 2000] C Henrik Westerberg et John Levine. *"GenPlan" : Combining Genetic Programming and Planning.* In Max Garagnani, editeur, 19^{th} Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2000), The Open University, Milton Keynes, UK, 14-15 Décembre 2000. 9, 39, 40

[Wolfman 1999] Steven A. Wolfman et Daniel S. Weld. *The LPSAT Engine and its Application to Resource Planning.* In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), pages 310–316, Stockholm, Sweden, Juillet 31-Août 6 1999. Morgan Kaufmann. 28

[Yu 2008] Tina Yu et Lawrence Davis. *An Introduction to Evolutionary Computation in Practice.* In Tina Yu, Lawrence Davis, Cem M. Baydar et Rajkumar Roy, editeurs, Evolutionary Computation in Practice, volume 88 of *Studies in Computational Intelligence*, pages 1–8. Springer, 2008. 67

[Yuan 2004] Bo Yuan et Marcus Gallagher. *Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms.* In Parallel Problem Solving from Nature - PPSN VIII, LNCS 3242, pages 172–181. Springer Verlag, 2004. 74

[Yuan 2007] Bo Yuan et Marcus Gallagher. Parameter setting in evolutionary algorithms, chapitre Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks, pages 121–142. 2007. 68, 70, 74