Divide-and-Evolve : une nouvelle méta-heuristique pour la planification temporelle indépendante du domaine

Marc Schoenauer
Projet TAO, INRIA Futurs, LRI
Bt. 490, Université Paris Sud
91405 Orsay
marc@lri.fr

Pierre Savéant
Thales Research & Technology
RD 128
91767 Palaiseau
pierre.saveant@thalesgroup.com

Vincent Vidal
CRIL - Université d'Artois
rue de l'université - SP16
62307 Lens
vidal@cril.univ-artois.fr

Résumé

Une approche originale dénommée *Divide-and-Evolve* est proposée pour l'hybridation des Algorithmes Évolutionnaires (AEs) avec des méthodes d'Intelligence Artificielle dans le domaine des Problèmes de Planification Temporelle (PPTs). Alors que les algorithmes mémétiques standards utilisent des méthodes locales de résolution pour améliorer les solutions évolutionnaires, l'approche *Divide-and-Evolve* divise arbitrairement le problème en plusieurs sous-problèmes (que l'on espère plus faciles), et peut ainsi résoudre globalement des problèmes hors d'atteinte lorsque directement fournis en entrée d'algorithmes spécialisés classiques. Mais le principal avantage de l'approche *Divide-and-Evolve* est qu'elle ouvre immédiatement une avenue pour l'optimisation multi-objectifs, même avec une méthode spécialisée mono-objectif. La preuve du concept de cette approche sur le benchmark de transport standard Zeno (mono-objectif) est donnée, et un petit benchmark multi-objectifs original est proposé dans ce même cadre Zeno pour montrer les possibilités multi-objectifs de la méthodologie proposée, une percée dans la planification temporelle.

1 Introduction

La *Planification Temporelle en Intelligence Artificielle* est une forme de résolution générale de problèmes se concentrant sur les problèmes pouvant s'exprimer dans des *modèles d'états* pouvant être définis par un espace d'états S, un état initial $s_0 \subseteq S$, un ensemble d'états buts $S_G \subseteq S$, un ensemble d'actions $A(s)$ applicables dans chaque état s, et une fonction de transition $f(a, s) = s'$ avec $a \in A(s)$, et $s, s' \in S$. Une solution à cette classe de modèles est une séquence d'actions applicables transformant l'état initial s_0 en un état but appartenant à S_G.

Une importante classe de problèmes est couverte par la Planification Temporelle qui étend la planification classique par l'ajout de durées aux actions en autorisant les actions concurrentes dans le temps [8]. D'autres métriques sont usuellement requises pour les problèmes de la vie réelle pour représenter la qualité d'un plan, par exemple un critère de coût ou de risque. Un technique habituelle est d'agréger les différents critères, mais cela est basé sur des aspects dépendants du problème et n'a pas forcément beaucoup de sens. Une meilleure solution est de calculer l'ensemble des solutions optimales non dominées – aussi appelé le front de Pareto.

À cause de la grande complexité combinatoire et des aspects multi-objectifs des Problèmes de Planification Temporelle (PPTs), les Algorithmes Évolutionnaires (AEs) [5] sont de bons candidats de méthodes généralistes. Basés sur un principe Darwinien simplifié à l'extrême – l'adaptation des espèces résulte de l'interaction entre la *sélection naturelle* et les *variations aveugles* survenant lors de la reproduction – les AEs sont des méthodes d'optimisation stochastique qui font évoluer une *population* d'*individus* (un ensemble de points de l'espace de recherche) en leur appliquant successivement des opérateurs de sélection, favorisant les points de meilleure *fitness* (valeur de la fonction objectif) et des opérateurs de *variation aveugle* (génération de nouveaux points à partir de points existants par modification aléatoires, indépendamment de leur fitness). Le succès de ces algorithmes, attesté par de nombreuses réussites sur des problèmes réels difficiles, provient essentiellement de leur très grande flexibilité : les AEs peuvent travailler sur des espaces de recherche très variés, du moment que l'on peut y définir des opérateurs de variation sématiquement liés au problème que l'on cherche à résoudre – comme la suite de cet article le montrera.

Cependant, il n'y a eu que très peu de tentatives d'appliquer des Algorithmes Évolutionnaires aux problèmes de planification, et à notre connaissance, aucune pour la Planification Temporelle. Certaines approches utilisent une représentation spécifique [15]. La plupart des approches indépendantes du domaine voient un plan comme un programme et utilisent la Programmation Génétique et le traditionnel Domaine des Cubes pour les expérimentations (depuis le "Genetic Planner" [17]). Un état de l'art plus complet sur la Planification Génétique peut être trouvé dans [1], où les auteurs ont expérimenté une représentation sous forme de chromosomes de longueur variable. Il est important de remarquer que tous ces travaux effectuent une recherche dans les espaces de plans partiels.

Il est aussi bien connu que les Algorithmes Évolutionnaires (AEs) peuvent rarement résoudre efficacement des Problèmes d'Optimisation Combinatoire par leur seules capacités, c'est-à-dire sans être hybridés, d'une façon ou d'une autre, avec des techniques de résolution locales *ad hoc*. Les plus réussies parmi ces hybridations utilisent les méthodes de Recherche Opérationnelle pour améliorer localement la descendance née des opérateurs de variation de l'AE (croisement et mutation) : ces algorithmes ont été appelés "Algorithmes Mémétiques" ou "Recherche Génétique Locale" [14]. Ces méthodes sont maintenant devenues l'objet d'un domaine de recherche à part entière, comme en témoignent la série des WOMA (*Workshops on Memetic Algorithms*) organisés chaque année, ainsi que plusieurs éditions spéciales de revues et livres édités (voir le récent [10]).

Cependant, la plupart des approches mémétiques sont basées sur la recherche d'améliorations locales de solutions candidates proposées par le mécanisme de recherche évolutionnaire, en utilisant des méthodes locales de résolution devant aborder le problème complet. Dans certains domaines combinatoires comme la Planification Temporelle, cela s'avère tout simplement impossible à partir d'un certain niveau de complexité des problèmes.

Cet article propose la méthode *Divide-and-Evolve*, inspirée du paradigme Divide-and-Conquer pour de telles situations : le problème est coupé arbitrairement en une séquence de sous-problèmes que l'on espère plus faciles à résoudre par une méthode locale spécialisée. La solution au problème original est alors obtenue en concaténant les solutions des différents sous-problèmes.

La section suivante présente une formulation abstraite du schéma *Divide-and-Evolve* depuis sa racine historique (et pédagogique), le paradigme TGV. Une représentation générique des opérateurs de variation est alors introduite. La section 3 présente une instanciation actuelle du schéma *Divide-and-Evolve* aux PPTs. Le cadre formel des PPTs est d'abord introduit, puis les aspects spécifiques aux PPTs sont présentés et discutés. La Section 4 est dévouée aux expérimentations sur le benchmark de transport Zeno pour les cas mono- et multi-objectifs. La dernière section ouvre une discussion exhibant les limitations de l'approche actuelle et donne quelques pistes actuelles et futures de poursuite de ces travaux.

2 Le paradigme *Divide-and-Evolve*

2.1 La métaphore TGV

La stratégie *Divide-and-Evolve* est née d'une métaphore sur des problèmes de planification de chemin pour le 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 [4], basé sur le fait que la seule méthode de résolution locale était un algorithme glouton pouvant seulement résoudre des problèmes très simples (i.e. sur de courtes distances). L'algorithme évolutionnaire recherche un découpage du chemin global en courts segments consécutifs tels qu'un algorithme local puisse facilement trouver un chemin joignant leurs extrémités. Les individus représentent les ensembles de stations de train intermédiaires entre la station de départ et le terminus. La convergence vers une bonne solution est obtenue avec la définition d'opérateurs de variation et de sélection appropriés [4]. Ici, l'espace d'états est la surface sur laquelle le trajectoire du train est définie.

Généralisation. Appliqué à la planification, le chemin est remplacé par une séquence d'actions et les "stations" deviennent des états intermédiaires du système. Le problème est ainsi divisé en sous-problèmes et "de courtes distances" deviennent "faciles à résoudre" par un algorithme local de résolution $\mathcal{L}$. L'algorithme évolutionnaire joue le rôle d'un oracle désignant des états par lesquels il est impératif de passer.

2.2 Représentation

Le problème considéré est le problème abstrait de la planification en Intelligence Artificielle comme décrit en introduction. La représentation utilisée par l'algorithme évolutionnaire est une liste de longueur variable d'états : un individu est ainsi défini par $(s_i)_{i\in[1,n]}$, où la longueur n est inconnue et sujette à évolution. Les états s_0 et $s_{n+1} \equiv s_G$ représentent l'état initial et le but du problème de départ, mais ne seront pas encodés dans les génotypes. En référence au paradigme TGV originel, chacun des états s_i d'un individu sera appelé une *station*.

Besoins. Le problème TGV originel est purement topologique, sans dimension temporelle, et se réduit à un problème de planification à une seule action : le déplacement entre deux points. La généralisation à un domaine de planification donné nécessite d'être capable de :

1. définir une distance entre deux états différents du système, telle que $d(s,t)$ soit autant que possible liée à la difficulté pour l'algorithme local $\mathcal{L}$ de trouver un plan solution conduisant de l'état initial s à l'état final t ;
2. générer une séquence chronologique de "stations" virtuelles, i.e. des états intermédiaires du système, proches les uns des autres, s_i étant proche de s_{i+1} ;
3. résoudre les problèmes "faciles" qui en résultent par l'algorithme local $\mathcal{L}$;
4. "combiner" les sous-plans en un seul plan résolvant le problème de départ.

2.3 Opérateurs de variation

Cette section décrit plusieurs opérateurs de variation pouvant être définis pour l'approche générique *Divide-and-Evolve* indépendamment du domaine d'application (par ex. les PPTs, ou le problème TGV originel).

Croisement. Les opérateurs de croisement s'attachent à échanger des stations entre deux individus. À cause de la nature séquentielle de la fitness, il semble une bonne idée d'essayer de préserver les séquences d'action, résultant en des adaptations immédiates de la représentation en longueur variable des opérateurs de croisement classiques 1- et 2-point.

Supposons que l'on veuille recombiner des individus $(s_i)_{1\leq n}$ et $(t_i)_{1\leq m}$. Le croisement 1-point consiste à choisir une station de chaque individu, par exemple s_a et t_b, et d'échanger la deuxième partie de leur liste de stations, obtenant deux descendants $(s_1, \ldots, s_a, t_{m+1}, \ldots t_b)$ et $(t_1, \ldots, t_b, s_{n+1}, \ldots, s_n)$ (le croisement 2-point est facilement implémenté, de façon similaire). Remarquons que dans les deux cas, la longueur de chaque descendant sera probablement différente de celle de ses parents.

Le choix des points de croisement s_a et t_b peut être soit uniforme (comme fait dans les travaux présentés ici), ou basé sur la distance, si une distance est disponible : choisir la première station s_a au hasard, puis choisir t_b par exemple par un tournoi basé sur la distance avec s_a (travail en cours).

Mutation. Plusieurs opérateurs de mutation peuvent être définis. Supposons que l'individu $(s_i)_{1\leq n}$ doive subir une mutation :

– **Au niveau de l'individu**, la mutation *Add* insère simplement une nouvelle station s_{new} après une station donnée s_a, résultant en une liste de longueur $n+1$: $(s_1, \ldots, s_a, s_{new}, s_{a+1}, \ldots, s_n)$. Sa contrepartie, la mutation *Del*, enlève une station s_a de la liste.
  Plusieurs améliorations sur le choix purement uniforme de s_a peuvent être ajoutées et font aussi partie d'un travail en cours : dans le cas où l'algorithme local échoue à faire joindre toutes les paires de stations successives, la dernière station qui a été atteinte avec succès par l'algorithme local peut être préférée pour être la station s_a (pour les deux mutations *Add* et *Del*). Si tous les problèmes partiels sont résolus, le plus difficile (par ex. en nombre de backtracks) peut être choisi.

– **Au niveau de la station**, la définition de chaque station peut être modifiée – mais ceci est dépendant du problème. Cependant, en supposant qu'il existe un opérateur de mutation de station μ_S, il est facile de définir la mutation de l'individu M_{μ_S} qui va simplement appeler μ_S sur chaque station s_i avec une probabilité définie par l'utilisateur p_{μ_S}. Des exemples d'opérateurs μ_S seront donnés en section 3, tandis qu'une simple mutation gaussienne des coordonnées (x, y) d'une station ont été utilisées pour le problème TGV originel [4].

3 Planification temporelle

3.1 Problèmes de planification temporelle

Les planificateurs indépendants du domaine utilisent en général le langage de description PDDL [13], hérité du modèle STRIPS [6], pour représenter un problème de planification. Ce langage est utilisé en particulier pour la compétition IPC[1] qui a lieu tous les deux ans depuis 1998. Ce langage a été étendu dans sa version 2.1 pour représenter des problèmes de planification temporelle. Pour des raisons de simplicité, le modèle temporel que nous utilisons ne se conforme pas strictement à PDDL2.1 [18].

Un *Opérateur PDDL temporel* est un tuple $o = \langle pre(o), add(o), del(o), dur(o)\rangle$ où $pre(o)$, $add(o)$ et $del(o)$ sont des atomes de base qui dénotent respectivement les préconditions, ajouts et retraits de o, et $dur(o)$ est un nombre rationnel qui représente la *durée* de o. Les opérateurs dans une description PDDL peuvent être décrits avec des variables utilisées dans les prédicats, comme `(at ?plane ?city)`.

Un *problème de planification temporelle (PPT)* est un tuple $P = \langle A, I, O, G\rangle$, où A est un ensemble d'atomes représentant tous les faits possibles d'une situation du monde, I et G deux ensembles d'atomes qui représentent respectivement l'état initial et le but du problème, et O un ensemble d'opérateurs PDDL instanciés.

De façon classique en planification dans les espaces de plans partiels (planification POCL) [19], deux actions fictives sont aussi considérées : $Start$ et End, chacune avec une durée nulle, la première sans précondition et produisant les atomes de l'état initial et la seconde sans effets et ayant les buts du problème en précondition.

Deux actions interfèrent quand l'une retire une précondition ou un ajout de l'autre. Le modèle temporel simplifié que nous utilisons [16] interdit à deux actions interférentes d'un plan valide de se chevaucher dans le temps. En d'autres termes, les préconditions d'une action doivent rester valides durant toute la durée d'exécution d'une action, et ses effets sont valides à la fin de son exécution mais ne peuvent être retirés par une action concurrente.

Un *ordonnancement* P est un ensemble fini d'occurrences d'actions $\langle a_i, t_i\rangle$, $i = 1, \ldots, n$, où a_i est une action et t_i est un rationnel positif indiquant la date de début de a_i (sa date de fin est $t_i + dur(a_i)$). P doit inclure les actions $Start$ et End, la première à l'indice de temps 0. Une même action (excepté ces deux dernières) peut être exécutée plusieurs fois dans P si $a_i = a_j$ pour $i \neq j$. Deux occurrences d'action a_i et a_j *se chevauchent* dans P si l'une démarre avant que l'autre finisse, c'est-à-dire si $[t_i, t_i + dur(a_i)] \cap [t_j, t_j + dur(a_j)]$ contient plus d'une unité de temps.

Un ordonnancement P est un *plan valide* ssi les actions interférentes ne se chevauchent pas, et pour chaque occurrence d'action $\langle a_i, t_i\rangle$ dans P ses préconditions $p \in pre(a_i)$ sont vraies au temps t_i. Cette condition est définie inductivement ainsi : p est vrai au temps $t = 0$ ssi $p \in I$, et p est vrai au temps $t > 0$ si soit p est vrai au temps $t-1$ et aucune action a dans P se terminant en t ne retire p, ou une action a' dans P se terminant en t ajoute p. La durée totale d'exécution (ou *makespan*) d'un plan P est l'indice de temps de l'action End.

[1] `http ://ipc.icaps-conference.org/`

TAB. 1 – Décomposition en stations de l'instance Zeno14 (la disposition des objets déplacés apparaît en gras.)

Objets	**Init** (station 0)	**Station 1**	**Station 2**	**Station 3**	**Station 4**	**But** (station 5)
plane 1	city 5	**city 6**	city 6	city 6	city 6	city 6
plane 2	city 2	city 2	**city 3**	city 3	city 3	city 3
plane 3	city 4	city 4	city 4	**city 9**	city 9	city 9
plane 4	city 8	city 8	city 8	city 8	**city 5**	city 5
plane 5	city 9	city 9	city 9	city 9	city 9	**city 8**
person 1	city 9	city 9	city 9	city 9	city 9	city 9
person 2	city 1	city 1	city 1	city 1	city 1	**city 8**
person 3	city 0	city 0	**city 2**	city 2	city 2	city 2
person 4	city 9	city 9	city 9	**city 7**	city 7	city 7
person 5	city 6	city 6	city 6	city 6	city 6	**city 1**
person 6	city 0	**city 6**	city 6	city 6	city 6	city 6
person 7	city 7	city 7	city 7	city 7	**city 5**	city 5
person 8	city 6	city 6	city 6	city 6	city 6	**city 1**
person 9	city 4	city 4	city 4	city 4	**city 5**	city 5
person 0	city 7	city 7	city 7	**city 9**	city 9	city 9
Makespan		350	350	280	549	522
Backtracks		1	0	0	195	32
Temps de recherche		0,89	0,13	0,52	4,34	1,64
Temps CPU total		49,10	49,65	49,78	54,00	51,83

	Compression	**Recherche globale**
Makespan	772	476
Backtracks	0	606405
Temps de recherche	0,01	4155,41
Temps CPU total	0,02 (total : 254,38)	4205,40

3.2 CPT : un planificateur temporel optimal

Un planificateur temporel optimal calcule un plan valide de makespan minimal. Même si l'utilisation d'un planificateur optimal n'est pas nécessaire (voir la discussion en section 5), nous avons choisi d'utiliser CPT [18], un planificateur temporel optimal librement accessible, pour sa dimension temporelle et pour son approche basée sur les contraintes qui procure des structures de données très utiles pour raccorder des solutions partielles (voir la section 2.2). En effet, comme en planification temporelle les actions peuvent se chevaucher dans le temps, une simple concaténation des sous-plans, bien que produisant une solution valide, va évidemment donner un makespan très éloigné de l'optimal, même si les sous-plans sont optimaux. Cependant, grâce aux liens causaux et aux contraintes d'ordre maintenus par CPT, un plan global amélioré peut être obtenu en décalant les sous-plans ou certaines actions les constituant au plus tôt dans le plan.

3.3 Rationale

Les raisons de l'échec des méthodes standard pour les PPTs vient de la complexité exponentielle du nombre de combinaisons d'actions possibles quand le nombre d'objets impliqués dans le problème augmente. Il est connu depuis longtemps que la prise en compte des interactions entre sous-buts peut diminuer la complexité de la recherche d'un plan, en particulier quand ces sous-buts sont indépendants [12]. De plus, calculer un ordre idéal sur ces sous-buts est aussi difficile que de trouver un plan (PSPACE-difficile), comme démontré dans [11]. L'idée de base pour l'utilisation de l'approche *Divide-and-Evolve* est que chaque sous-plan local ("joignant" les stations s_i et s_{i+1}) devrait être plus facile à trouver que le plan global (joignant la station de départ s_0 et le terminus s_{n+1}). Nous allons maintenant montrer cette intuition sur le benchmark Zeno, qui permet de modéliser des problèmes de transport (voir `http ://ipc.icaps-conference.org/`).

La Table 1 illustre la décomposition d'un problème relativement difficile du domaine Zeno (`zeno14` venant des benchmarks d'IPC-3), un problème de transports avec 5 avions (`plane1` à `plane5`) et 10 personnes (`person0` à `person9`) voyageant entre 10 villes (`city0` à `city9`).

En analysant la solution optimale trouvée par CPT-3 il a été facile de diviser le "chemin" optimal de cette solution dans l'espace des états en quatre stations intermédiaires entre l'état initial et le but. On peut constater que très peu mouvements (avions ou personnes) ne se passent entre deux stations consécutives (en gras dans la Table 1). Les sous-plans sont trouvés facilement par CPT, avec un maximum de 195 backtracks et 4,34 secondes de temps de recherche. On peut remarquer que CPT passe la majorité de son temps en pré-processing : cette opération est répétée à chaque exécution de CPT, et pourrait être factorisée à peu de frais.

L'étape finale du procédé est la compression des cinq sous-plans (voir la section 2.2) : elle est effectuée en 0,02 secondes sans aucun backtrack, donnant un makespan global de 772, largement inférieur à la somme des makespan de chaque sous-plan (2051).

En résumé, le plan recomposé, avec un makespan de 772, requiert un temps de calcul total de 254,38 secondes (comprenant seulement 7,5s de recherche pure) et 228 backtracks, alors que le plan optimal de makespan 476 est trouvé par CPT en 4205 secondes et 606405 backtracks.

Ce phénomène sera discuté en section 5.

3.4 Description de l'espace d'états

États non temporels. Un espace d'états naturel pour les PPTs, comme décrits au début de cette section, serait l'espace actuel de tous les états indexés par le temps du système. Évidemment, la taille d'un tel espace est bien trop grande, nous l'avons donc simplifié en se restreignant aux états non temporels. Cependant, même avec cette simplification, tous les états "non temporels" ne peuvent être considérés dans la description des "stations".

Limiter les états possibles. Premièrement, l'espace des états grandit de façon exponentielle avec la taille des problèmes. Deuxièmement, tous les états ne sont pas consistants avec le domaine de planification. Ainsi, un objet ne peut se trouver en deux endroits différents au même instant dans un problème de transport – et inférer un tel invariant d'état, bien que faisable, n'est pas trivial [7]. De plus, le problème de l'existence d'un plan pour une description STRIPS propositionnelle a été démontré comme étant PSPACE-complet [2].

Un contournement possible de cette difficulté serait d'utiliser un algorithme local pour (rapidement) vérifier la consistance d'une situation donnée, et pénaliser les stations inatteignables. Cependant, cela constituerait clairement une perte de ressources de calcul.

D'un autre côté, l'introduction de connaissances sur le domaine dans les algorithmes évolutionnaires est depuis longtemps connu comme la route royale vers le succès en Calcul Évolutionnaire [9]. Ainsi, il semble plus prometteur d'ajouter les invariants d'états à la description de l'espace de recherche pour supprimer autant d'états inconsistants que possible. Un avantage est qu'il n'est pas nécessaire de supprimer *tous* les états inconsistants puisque, dans tous les cas, l'algorithme local est là pour aider l'AE à les éliminer – les stations inconsistantes auront une mauvaise fitness, et ne survivront pas aux étapes de sélection suivantes. En particulier, seuls les invariants d'état impliquant un seul prédicat sont actuellement considérés.

3.5 Représentation des stations

Il a ainsi été décidé de décrire les stations en utilisant *seulement les prédicats présents dans les buts du problème global*, et de maintenir des invariants d'état basés sur la sémantique du problème.

Un bon exemple est donné dans la table 1 : le but de cette instance est de déplacer des personnes et des avions dans les villes de la dernière colonne. Aucun autre prédicat que les `(at objectN cityM)` correspondants n'est présent dans le but. À l'aide d'un fichier fourni par l'utilisateur, l'algorithme sait que seuls les prédicats `at` seront utilisés pour la représentation des stations, le premier argument de `at` ne pouvant apparaître qu'une seule fois (`at` est dit *exclusif* par rapport à son premier argument). L'espace d'états qui sera exploré par l'algorithme se réduit ainsi à un vecteur de 15 atomes dénotant qu'un objet se trouve dans une ville donnée (une colonne de la table 1). En addition, l'implémentation actuelle d'une station inclut la possibilité de supprimer un prédicat de la liste : l'objet correspondant ne sera alors pas déplacé par le sous-plan.

Distance. La distance entre deux stations doit refléter la difficulté, pour l'algorithme local, de trouver un plan qui les fasse se rejoindre. Pour l'instant, une distance purement syntaxique et indépendante du domaine est utilisée : le nombre de prédicats différents non encore atteints. La difficulté peut alors être estimée par le nombre de backtracks nécessaires à l'algorithme local. Il est raisonnable de supposer qu'ainsi, la plupart des problèmes où peu de prédicats doivent être modifiés entre l'état initial et le but seront faciles pour l'algorithme local – bien que cela ne soit certainement pas vrai en général.

3.6 Opérateurs de variation spécifiques

Initialisation. D'abord, le nombre de stations est choisi de façon uniforme dans un intervalle fourni par l'utilisateur. Ce dernier donne aussi une distance maximale d_{max} entre les stations. Une matrice est alors construite, similaire à la table 1 : chaque ligne correspond à un des prédicats but, chaque colonne étant une station. Seules les premières et dernières colonnes (correspondant à l'état initial et au but) sont initialisées. Un certain nombre de "déplacements" est alors effectué au hasard dans la matrice, au plus d_{max} par colonne, et au moins un par ligne. Des déplacements additionnels sont alors ajoutés au moyen d'un autre paramètre fourni par l'utilisateur, sans dépasser la limite d_{max} par colonne. La matrice est alors totalement remplie, en partant des deux extrémités (init et but), contrainte dans chaque colonne par les invariants d'état. Un passage final sur tous les prédicats de chaque station en supprime un certain nombre avec une probabilité donnée.

Mutation des stations. Grâce à la représentation simplifiée des états (un vecteur de fluents avec des invariants d'états), il est immédiat de modifier une station aléatoirement : avec une probabilité donnée, une nouvelle valeur pour les arguments non exclusifs est choisie parmi les valeurs possibles respectant toute les contraintes (y compris les contraintes de distance entre les stations précédente et suivante). De plus, chaque prédicat peut être enlevé d'une station avec une probabilité donnée, comme dans la phase d'initialisation.

4 Premières expérimentations

4.1 Optimisation mono-objectif

Notre terrain favori pour la validation de l'approche *Divide-and-Evolve* est celui des problèmes de transport, et a débuté par le domaine *Zeno* décrit dans la section 3.3. Comme on peut le voir dans la table 1, la description des stations dans le domaine `Zeno` implique un seul prédicat, `at`, avec deux arguments. Il est *exclusif* par rapport à son premier argument. Trois instances ont été testées, nommées zeno10, zeno12 et zeno14, par ordre croissant de difficulté.

Réglages algorithmiques. L'algorithme AE de la première implémentation du paradigme *Divide-and-Evolve* utilise les réglages algorithmiques standards au niveau de la population : un moteur d'évolution de type $(10, 70)-ES$ (10 parents donnent naissance à 70 enfants, dont les 10 meilleurs deviennent les parents suivants), les enfants sont créés en utilisant 25% de croisements 1-point et 75% de mutations, dont la moitié est une mutation *Add* ou *Del* et l'autre moitié une mutation de station dépendante du problème : un prédicat est modifié aléatoirement avec probabilité 0.75 et est supprimé (resp. restauré) avec probabilité 0.125 (voir section 3.6). L'initialisation est effectuée avec une taille initiale dans $[2, 10]$, une distance maximale de 3 et une probabilité de retirer un prédicat de 0,1. Il est à noter que pour ces premières expérimentations visant à apporter la preuve du concept, les valeurs ci-dessus ont été décidées après un nombre très limité d'expérimentations initiales et qu'il est certainement possible d'améliorer ces réglages.

La fitness. L'objectif principal est ici le makespan total du plan – en supposant qu'un plan global puisse être trouvé, c'est-à-dire que tous les problèmes (s_i, s_{i+1}) puissent bien être résolus par l'algorithme local. Dans le cas où un problème local ne peut pas être résolu, l'individu est déclaré infaisable et est pénalisé de façon à ce que tous les individus infaisables soient pires que n'importe quel individu faisable. De plus, cette pénalité est proportionnelle au nombre de stations restant à résoudre après l'échec, afin créer un "gradient" emmenant l'algorithme vers des solutions faisables.

Pour les individus faisables, une moyenne entre le makespan total et la somme des makespan des plans des problèmes partiels est utilisée : quand seul le makespan total est utilisé, certains individus voient leur nombre de station augmenter démeusurément (par l'addition par exemple de suites d'allers-retours d'un objet entre 2 villes), sans conséquence sur le makespan total grâce à la compression finale effectuée par CPT, mais ralentissant malgré tout l'exécution dans son ensemble à cause de la répétition d'appels inutiles à CPT. Ce phénomène, classique dans les représentations de longueur variable, doit donc être évité pour des raisons d'efficacité.

Résultats préliminaires. Les instances simples zeno10 (resp. zeno12) peuvent être facilement résolues par CPT-2 seul, en moins de 2s (resp. 125s), en trouvant le plan optimal de makespan 453 (resp. 549) avec 154 (resp. 27560) backtracks.

Pour zeno10, toutes les exécutions de *Divide-and-Evolve* trouvent la solution optimale dans les toutes premières générations (i.e. la procédure d'initialisation produit toujours un individu faisable que CPT peut compresser au makespan optimal). Pour zeno12, toutes les exécutions trouvent une solution suboptimale avec un makespan compris entre 789 et 1222. On peut noter que la solution finale a été trouvée après 3 à 5 générations, l'algorithme restant fixé sur cette solution par la suite. Le temps CPU requis pour 10 générations tourne autour de 5 heures.

Un cas plus intéressant est celui de zeno14. Tout d'abord, cela vaut la peine de mentionner que l'AE *Divide-and-Evolve* actuel utilise l'algorithme local CPT version 2, et que cette version est incapable de trouver une solution à zeno14 : le résultat donné dans la table 1 a été obtenu en utilisant une nouvelle version de CPT encore expérimentale et pas utilisable directement depuis l'AE. Mais alors que CPT-2 n'a pu résoudre le problème global, il a cependant pu être utilisé pour résoudre les petites instances de zeno14 générées par l'approche *Divide-and-Evolve* – prenant cependant un temps CPU prohibitif. Fixer une limite sur le nombre de backtracks autorisés pour CPT a été aussi obligatoire pour forcer CPT à ne pas explorer les cas trop complexes qui auraient résulté en des appels bien trop longs.

Cependant, un individu faisable a été trouvé pour chacun des deux seules exécutions que nous avons pu faire – le calcul d'une génération (70 évaluations) prenant plus de 10 heures. Pour la première, un individu faisable a été trouvé dans la population initiale, de makespan 1958, la meilleure solution ayant un makespan de 773. Dans l'autre exécution, la première solution faisable a été trouvée à la troisième génération – mais l'algorithme n'a jamais pu améliorer cet individu (de makespan 1356).

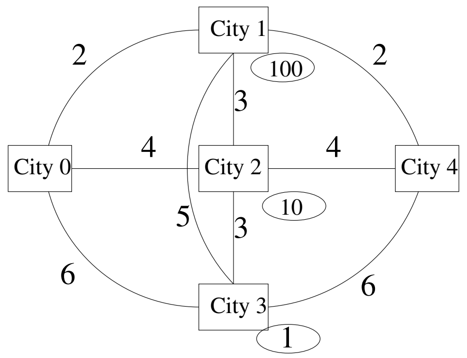
Bien que décevants par rapport aux performances globales de l'algorithme, ces résultats témoignent cependant du fait que l'approche *Divide-and-Evolve* peut effectivement résoudre un problème inabordable pour CPT seul (compte tenu du fait que la version de CPT utilisée pour toutes les expérimentations est de loin moins efficace que celle utilisée pour résoudre zeno14 dans la section 3.3, et se trouve incapable de résoudre zeno14).

4.2 Un problème multi-objectifs

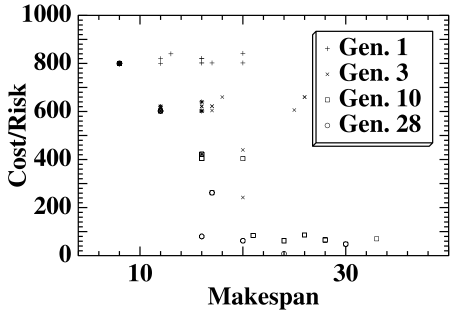
Description du problème. Afin de tester la faisabilité d'une approche multi-objectifs basée sur le paradigme *Divide-and-Evolve*, nous avons étendu le domaine zeno avec un critère supplémentaire, qui peut être interprété comme un coût, ou un risque : dans le premier cas, cet objectif additionnel est une mesure additive, dans le second cas la fonction d'agrégation est l'opérateur max.

L'instance du problème est montrée dans la Figure 1 : les chemins possibles entre les villes sont représentés par des arcs, il n'existe qu'une seule méthode de transport (l'avion), et chaque arc est étiqueté par la durée du transport correspondant. Les risques (ou coûts) sont attachés aux villes (i.e., concernent tous les moyens de transport qui décollent ou atterrissent dans ces villes). Dans l'état initial, les trois personnes et les deux avions sont en City 0, le but étant de les transporter en City 4.

Comme il peut être facilement calculé, il y a trois solution pareto-optimales remarquables, correspondant à la traversée d'une seule des trois cités médianes. Traverser



a) L'instance : les durées sont attachées aux arcs, les coûts/risques sont attachés aux villes (cercles).



b) La population à différentes générations pour une exécution réussie sur l'instance avec coût (additif) du mini-problème `Zeno` de la Figure 1-a.

FIG. 1 – Le benchmark Zeno multi-objectifs.

`City 1` est rapide, mais risqué (coûteux), alors que passer par `City 3` est lent et sûr (peu coûteux).
Quand toutes les personnes traversent respectivement `City 1`, `City 2` et `City 3`, les valeurs correspondantes des makespan et des coûts dans le cas additif sont `(8, 800)`, `(16, 80)` et `(24, 8)`, alors qu'elles sont, dans le cas du max, `(8, 100)`, `(16, 10)` et `(24, 1)`.

Complexité du problème. Il est facile de calculer le nombre de stations virtuelles possibles : chacune des 3 personnes peut être dans une des 5 villes, ou dans un endroit indéfini (absence du prédicat correspondant). Ainsi, il y a $3^6 = 729$ combinaisons possibles, et 729^n listes possibles de longueur n. Donc même si n est limité à 6, la taille de l'espace de recherche est approximativement 10^{17} ...

L'algorithme. L'AE est basé sur l'algorithme évolutionnaire standard NSGA-II multi-objectif [3] : tournoi de sélection standard de taille 2 et remplacement déterministe parmi les parents+descendance, les deux basés sur le rang de Pareto et la distance de surpeuplement ; une population de taille 100 évolue durant 30 générations. Tous les autres paramètres sont identiques au cas mono-objectif.

Fitnesses. Le problème comporte deux objectifs : l'un est le makespan total (comme dans le cas mono-objectif), l'autre est le **risque** (agrégé par l'opérateur `max`) ou le **coût** (un objectif **additif**). Comme le risque global ne peut prendre que 3 valeurs, il n'y a aucun moyen d'obtenir une information graduelle utile quand il est utilisé comme fitness dans le cas du max. Cependant, même dans le cas additif, le même argument que pour le makespan s'applique (section 4.1), et ainsi, dans tous les cas, le second objectif est la somme du risque/coût global et la moyenne (pas la somme) des valeurs pour les problèmes partiels – en excluant de cette moyenne les problèmes partiels qui ont un makespan nul (quand le but est inclus dans l'état initial).

Résultats. Dans le cas du **coût** (additif), l'optimum de Pareto le plus difficile (traverser seulement la ville 3) a été trouvé 4 fois sur 11 exécutions. Cependant, les deux autres optima de Pareto remarquables, ainsi que plusieurs autres points du front de Pareto, ont aussi été trouvés à chaque exécution. La Figure 1-b montre différents aperçus de la population à différentes étapes de l'évolution pour une exécution réussie typique : au départ ('+'), tous les individus ont un coût élevé (supérieur à 800) ; à la génération 3 ('×'), certains individus de la population ont un coût inférieur à 600 ; à la génération 10 (les carrés), beaucoup de points ont un coût inférieur à 100. Mais la solution optimale (24,8) est trouvée seulement à la génération 28 (les cercles).
Le problème dans le contexte du **risque** (le cas du max) s'est montré, comme prévu, légèrement plus difficile. Les trois optima de Pareto (il n'y a pas d'autre point sur le vrai front de Pareto dans le cas max) ont été trouvés dans deux exécutions sur 11. Cependant, toutes les exécutions ont trouvé les deux autre optima de Pareto, ainsi que la solution légèrement suboptimale consistant à traverser seulement `city 3` mais sans utiliser la petite astuce mentionnée plus haut, résultant en une solution (36,1).
Dans les deux cas, ces résultats valident clairement l'approche *Divide-and-Evolve* pour les PPTs multi-objectifs – CPT n'a aucune connaissance du risque/coût dans sa procédure d'optimisation - il agrège seulement les valeurs a posteriori, après avoir calculé son plan optimal basé seulement sur le makespan, d'où la difficulté qu'il a à trouver le point du front de Pareto ne passant que par `city3`, puisque ce point correspond aux trajets **les plus lents**.

5 Discussion et travaux futurs

Un souci principal est l'existence d'une décomposition pour tous les plans de makespan optimal. Malheureusement, à cause de la restriction sur la représentation la limitant aux prédicats du but, certains états deviennent impossibles à décrire. Si un de ces états est obligatoire pour l'obtention d'un plan optimal, l'algorithme évolutionnaire sera incapable de le trouver. Dans le benchmark `zeno14` détaillé en section 3.3 par exemple, on peut voir dans la solution optimale que le prédicat `in` devrait être pris en compte en effectuant la division de la solution optimale, afin d'être capable de lier une personne donnée avec un avion spécifique. La difficulté principale, cependant, est d'ajouter l'invariant d'état entre `at` et `in` (une personne est soit `at` un endroit, soit `in` un avion). Nous envisageons pour la suite de ce travail de prendre en compte les invariants d'état impliquant des paires des prédicats, afin de traiter de tels cas. Dans le même esprit, nous allons aussi tenter de voir si de tels invariants ne peuvent pas être déduits automatiquement des structures de données maintenues par CPT.

Il est clair d'après les résultats quelque peu décevants présentés dans la section 4.1 que les capacités de recherche de l'algorithme proposé devraient être améliorées ; et il y a beaucoup de possibilités d'améliorations. D'abord, le plus immédiat, les opérateurs de variation pourraient utiliser des connaissances dépendantes du domaine, comme proposé dans la section 2.3 – même si cela s'éloigne de la "pure" recherche évolutionnaire aveugle. Également, tous les paramètres de l'algorithme devraient être plus finement réglés.

Bien sûr, le schéma *Divide-and-Evolve* sera expérimenté sur d'autres exemples, comme ceux proposés par la compétition internationale de planification. Des résultats préliminaires dans le domaine `Driver` montrent des résultats très similaires à ceux du domaine `Zeno`. Mais bien d'autres domaines, comme `Depots`, impliquent (au moins) deux prédicats dans la description des buts (par ex., `in` et `on` dans `Depots`). Il est ainsi nécessaire d'augmenter la portée des expressions reconnues pour la description des individus.

D'autres améliorations viendront du passage à la nouvelle version de CPT, récrite entièrement en C, qu'il sera possible d'appeler directement depuis l'AE, n'effectuant ainsi qu'une seule fois l'ensemble des opérations de préprocessing : actuellement, CPT est relancé entièrement à chaque calcul d'une solution partielle, et un rapide regard à la table 1 montre que sur le problème `zeno14`, par exemple, le temps d'exécution par individu baissera de 250 à 55 secondes environ. Bien que cela n'améliorera pas *per se* la qualité des résultats, cela permettra de s'attaquer à des problèmes encore plus complexes que `zeno14`. Dans le même ordre d'idées, d'autres planificateurs, en particulier suboptimaux, peuvent aussi être employés à la place de CPT, car l'approche *Divide-and-Evolve* peut trouver des résultats optimaux en utilisant des planificateurs suboptimaux (comme cela a été fait ici dans le cas multi-objectifs).

Une dernière mais importante remarque à propos des résultats est que, au moins dans le cas mono-objectif, la meilleure solution trouvée par l'algorithme l'a toujours été dans les toutes premières générations des différentes exécutions : il est possible que le simple découpage d'un problème en sous-problèmes plus simples durant la phase d'initialisation soit la raison principale de ces résultats. Des investigations poussées montreront si oui ou non un Algorithme Évolutionnaire est utile dans ce contexte !

Malgré tout, nous pensons que l'utilisation du Calcul Évolutionnaire restera nécessaire pour résoudre les problèmes multi-objectifs, comme en témoignent les résultats de la section 4.2, qui sont, à notre connaissance, les tous premiers résultats d'optimisation de Pareto pour les PPTs.

Références

[1] A. H. Brie and P. Morignot. Genetic Planning Using Variable Length Chromosomes. In *15th Intl Conf. on Automated Planning and Scheduling*, 2005.

[2] T. Bylander. The Computational Complexity of Propositional STRIPS planning. *Artificial Intelligence*, 69(1-2) :165–204, 1994.

[3] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization. In M. Schoenauer et al., eds, *PPSN VI*, pp 849–858. Springer-Verlag, LNCS 1917, 2000.

[4] C. Desquilbet (Supervisor : M. Schoenauer) Détermination de trajets optimaux par algorithmes génétiques. Rapport de stage de l'Ecole Polytechnique, 1992.

[5] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*, Springer Verlag, 2003.

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

[7] M. Fox and D. Long. The Automatic Inference of State Invariants in TIM. *Journal of Artificial Intelligence Research*, 9 :367–421, 1998.

[8] H. Geffner. Perspectives on Artificial Intelligence Planning. In *AAAI-2002*, pp 1013–1023, 2002.

[9] J. J. Grefenstette. Incorporating Problem Specific Knowledge in Genetic Algorithms. In Davis L., ed., *Genetic Algorithms and Simulated Annealing*, pp 42–60. Morgan Kaufmann, 1987.

[10] W.E. Hart, N. Krasnogor, and J.E. Smith, editors. *Recent Advances in Memetic Algorithms*. Studies in Fuzziness and Soft Computing, Vol. 166. Springer Verlag, 2005.

[11] J. Koehler and J. Hoffmann. On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm. *JAIR*, 12 :338–386, 2000.

[12] R. Korf. Planning as Search : A Quantitative Approach. *Artificial Intelligence*, 33 :65–88, 1987.

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

[14] P. Merz and B. Freisleben. Fitness Landscapes and Memetic Algorithm Design. In David Corne et al., eds, *New Ideas in Optimization*, pp 245–260. McGraw-Hill, London, 1999.

[15] J.L. Schlabach, C.C. Hayes, and D.E. Goldberg. FOX-GA : A Genetic Algorithm for Generating and Analyzing Battlefield Courses of Action. *Evolutionary Computation*, 7(1) :45–68, 1999.

[16] D. Smith and D. S. Weld. Temporal Planning with Mutual Exclusion Reasoning. In *IJCAI-99*, pp 326–337, 1999.

[17] L. Spector. Genetic Programming and AI Planning Systems. In *AAAI 94*, pp 1329–1334. MIT Press, 1994.

[18] V. Vidal and H. Geffner. Branching and Pruning : An Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence*, 170(3) :298–335, 2005.

[19] D. S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 15(4) :27–61, 1994.