HAL
open science

Planification Evolutionnaire par Décomposition

Jacques Bibai, Marc Schoenauer, Pierre Savéant, Vincent Vidal

To cite this version:

Jacques Bibai, Marc Schoenauer, Pierre Savéant, Vincent Vidal. Planification Evolutionnaire par Décomposition. [Intern report] RT-0355, 2008. inria-00322880v1

HAL Id: inria-00322880

https://inria.hal.science/inria-00322880v1

Submitted on 18 Sep 2008 (v1), last revised 8 Oct 2008 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Evolutionary Planning Decomposition

Jacques Bibaï — Marc Schoenauer — Pierre Savéant — Vincent Vidal

N° ????

Septembre 2008

Thème COG

Rapport technique

ISSN 0249-0803 ISRN INRIA/RT--????--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche SACLAY - ÎLE-DE-FRANCE

Evolutionary Planning Decomposition

Jacques Bibaï* † , Marc Schoenauer * , Pierre Savéant † , Vincent Vidal ‡

Thème COG — Systèmes cognitifs
Équipe-Projet TAO

Rapport technique n° ???? — Septembre 2008 — 19 pages

Abstract: This paper presents a stochastic approach for domain-independent planning decomposition termed *Divide-and-Evolve*. The basic idea is to search the space of state decompositions of the planning problem at hand by means of Artificial Evolution: candidate solutions are sequences of intermediate goals which define consecutive planning subproblems that are hopefully easier than the global problem. We focus on simple temporal problems and our reference planner for solving the subproblems is CPT. The constraint-based representation of CPT facilitates the implementation of an efficient routine to compress the subplans in a global solution plan. We describe the individual representation, the variation operators as well as the fitness and report on parameter tuning. We have experimented our approach on several IPC benchmarks and compare first to CPT and then to the best results found by state-of-the-art planners. Results validate the concept and show quality improvement.

Key-words: Evolutionary Planning, temporal planning, state decompositions, Divide-and-Evolve,Planning Decomposition

* Projet TAO, INRIA Saclay & LRI, Université Paris Sud, Orsay, France, firstname.lastname@inria.fr
† Thales Research & Technology, Palaiseau, France, firstname.lastname@thalesgroup.com
‡ CRIL & Université d'Artois, Lens, France, vidal@cril.univ-artois.fr

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Planification Evolutionnaire par Décomposition

Résumé : Ce rapport présente l'approche *Divide-and-Evolve* pour la résolution générique des problèmes de planification temporelle par décomposition. L'idée principale de l'approche est la recherche des solutions dans l'espace des décompositions en états intermédiaires à l'aide d'un algorithme évolutionnaire: les solutions candidates sont des séquences d'états intermédiaires qui définissent successivement les plans partiels du problème initial. Nous nous sommes interessés à la résolution des problèmes de type "simple temporal planning problems". La résolution des séquences d'états intermédiaires et la détermination d'une solution globale se font à l'aide du planificateur CPT. Ce rapport formalise l'approche, définit l'algorithme Divide Evolve et compare les résultats obtenus à ceux trouvés par les meilleurs planificateurs existants à notre connaissance.

Mots-clés : programmation génétique, algorithme évolutionnaire, planification évolutionnaire, planification temporelle, Divide-and-Evolve, décomposition des problèmes de planification

Contents

1 Introduction **4**

2 Decomposition as an Optimization Problem **5**
2.1 Rationale 5
2.2 The search space 5
2.3 The objective function 6

3 CPT and subplan compression **7**
3.1 An optimal planner 7
3.2 Subplan compression 7

4 Divide-and-Evolve **9**
4.1 Choosing an Optimization Algorithm 9
4.2 Evolutionary Algorithms 9
4.3 Representation 10
4.3.1 Using predicate IN 10
4.3.2 Using predicate AT 10
4.4 Fitness 11
4.5 Variation operators 11
4.5.1 Crossover operator 12
4.5.2 Mutations 12
4.5.3 Applying variation operators 12
4.6 Initialization of the population 13

5 Experimental results **13**
5.1 The domains 13
5.2 The Reference Planners 13
5.3 *Divide-and-Evolve* settings 14
5.4 Results 14
5.4.1 Validating the *Divide-and-Evolve* approach 15
5.4.2 D&E versus LPG 15

6 Discussion **15**
6.0.3 Complexity 15
6.0.4 Embedding Other Planners 16
6.0.5 Parameter tuning 16

7 Conclusion **16**

1 Introduction

Applying a *Divide-and-Conquer* strategy for solving planning problems has been investigated in various deterministic approaches. They usually exploit properties of the domain or problem structure in order to identify independent subproblems. In [12] for instance, the division is based on ordered landmarks [8] whereas in SGPlan [4], subgoal partitioning is based on mutual-exclusion constraints which show strong locality in most of IPC benchmarks. The objective of these methods is to reduce time complexity, though sometimes at the cost of a lower quality solution plan.

As stated in [12], any planning decomposition method requires (a) a decomposition principle, (b) an algorithm to solve the subproblems individually and (c) a procedure to recombine the subplans into a global solution plan.

This paper presents a stochastic approach for Planning Decomposition (i.e. addressing the (a) issue above) termed *Divide-and-Evolve* (D&E), deepening previous work from the same authors in an evolutionary context (not cited at the moment for anonymity reasons). The basic idea is to search the space of state decompositions of the planning problem at hand by means of Artificial Evolution: candidate solutions are sequences of intermediate goals which define consecutive planning subproblems that are hopefully easier than the global problem. The (b) issue, solving the individual subproblems, is addressed by handling the subproblems over to a standard planner. Though any planner could a priori be used, the choice of CPT [14] was motivated not only by its efficiency as an optimal temporal planner, but also because of its internal representation, that help solving the (c) issue, subplan recombination.

Indeed, when the nested planner succeeds in solving all subproblems, the concatenation of subplans is already a solution to the global problem. However, this solution is likely to be suboptimal since concurrency among actions might be lost. A *compression* step of such a naive concatenation is hence needed. Unfortunately, compressing a set of subplans into a global plan can be a difficult problem per se with respect to the expected level of compression. For example, finding the optimal global plan containing exactly all the actions of the subproblems without taking into account any other information from the subplans is NP-complete. However, as the sum of subplan makespans gives an upper bound for the global makespan, it is not PSPACE-complete, like propositional STRIPS planning [3]. Taking into account more constraints from the subplans degrades the quality of the global plan, but thanks to the constraint-based representation of CPT and its Partial Order Causal Link (POCL) planning strategy [11], an efficient compression routine that exploits causal links and precedence constraints can be implemented.

The D&E approach can be seen as an attempt to stochastically generate ordered sets of landmarks. However, the method is not limited to finding sets of facts that must absolutely be true within every solution to the initial problem. In particular, it also applies to problems that have no landmark per se, for simple symmetry reasons: there can be several equivalent candidate landmarks, and only one of them can and must be true at some point.

The first validation of the *Divide-and-Evolve* approach is to compare it to CPT alone: the goal of Planning Decomposition is to reduce the complexity of the problem. Further evidence of efficiency will be brought by experimental comparisons with some of the best-performing planners on IPC benchmarks.

However, several issues have to be taken into account when comparing *Divide-and-Evolve* with other planners: the most spectacular results are those where *Divide-and-Evolve* improves over the best known results for a given problem instance. But other measures of performance need to be considered, and depend on the target application: in a *design context*, where one planning problem has to be solved the best possible way, and multiple instances of the algorithm can be run, it is important to obtain the best possible makespan, even very rarely. On the other hand, in a *production context*, where different planning problems have to be solved rapidly and repeatedly, it is important to be able to produce good solutions reliably. Hence both the quality of the best plan obtained over multiple trials and the robustness of the algorithm (proportion of runs that obtain this best solution) will be considered when evaluating the *Divide-and-Evolve* approach.

2 Decomposition as an Optimization Problem

Our goal is to solve temporal planning problems given in PDDL form, focusing on simple grounded temporal problems, i.e. that extend basic STRIPS formulation by attaching a duration to each action. A *temporal planning problem* is a tuple $P = \langle A, O, I, G \rangle$, where A is a set of atoms representing all the possible facts in a world situation, O is a set of actions, and I and G are two sets of atoms that respectively denote the initial state and the problem goal. An *action* is a tuple $a = \langle pre(a), add(a), del(a), dur(a) \rangle$ where $pre(a)$, $add(a)$ and $del(a)$ are sets of ground atoms that respectively denote the preconditions, add effects and del effects of a, and $dur(a)$ denotes the *duration* of a, a rational number.

2.1 Rationale

The aim of Planning Decomposition is to transform a given planning problem into a sequence of hopefully easier subproblems. Indeed, temporal planning introduces concurrency among actions, involving in turn resource sharing issues. Among the multiple alternative plan trajectories, combinatorial explosion in temporal planning results for instance from task ordering permutations, that produce equivalent quality plans but are nevertheless searched for by most planners. The Ferry benchmark is a canonical example that illustrates this phenomenon: there is only one resource, and any task ordering is optimal. It is obvious here that almost any decomposition will lead to simpler subproblems, until the point where no multiple choice exist at all. At the opposite, when multiple resources can be used concurrently (or multiple actions can be driven concurrently), a strong decomposition might miss potential parallelism, that then need to be recovered by the compression stage.

2.2 The search space

A general decomposition is simply a sequence of intermediate partial states $(s_i)_{i \in [0,n+1]}$, where s_0 is the initial state, and s_{n+1} is the goal of the problem at hand, and, such that, for each $i \in [0, n]$, the planning problem whose initial state is s_i and final state is s_{i+1} can easily be solved by some standard planner.

Several issues need to be addressed, before deciding on the exact representation of such decomposition. First, what atoms should be selected to describe the intermediate states? Indeed, whereas the initial state is necessarily complete (i.e. it fully describes a world state), the goal is generally only given as a partial description. Hence it seems reasonable to restrict intermediate states to partial description as well, for the sake of complexity. This however raises the question of which atoms to choose.

Another crucial issue is that of the consistency of a given set of atoms: when exploring the space of possible intermediate states, it would be inefficient to try some states that are obviously inconsistent (e.g. two atoms involve the same object, and put it in different locations, in a typical logistic problem). However, a complete consistency check would involve solving the corresponding subproblem, so a trade-off has to be made.

Last, but not least, the only useful decompositions are those for which all resulting subproblems are simpler to solve than the initial problem, for the planner at hand. Unfortunately, the only way to ensure that would be to actually run the planner, as no sensible distance metric exists such that the distance between the initial state and the goal would give an indication about the problem difficulty. The workaround used for *Divide-and-Evolve* is to use a purely syntactic distance, even though it is clearly inappropriate. But the lack of a good estimate of the difficulty of a subproblem raises another issue: it can be the case that some subproblems from a given decomposition are as hard as the initial problem itself. The embedded planner will take an enormous amount of time to solve such subproblems, if it ever returns a result. It is mandatory to avoid such endless runs of the local planner, that would undermine the whole *Divide-and-Evolve* idea.

2.3 The objective function

An essential step for all optimization problems is the design of the objective function. But two very different situations should be distinguished here. First, when the embedded planner is able to find a solution to all subproblems, the combination of all subplans (or its compression into a shorter plan) is a solution of the original planning problem: its total makespan is the clear objective to be minimized.

However, when the embedded planner fails to solve one of the subproblems, such objective function makes no sense any more, and simply discarding such decompositions might result in a deadlock for the most complex problems where even finding a suboptimal plan is difficult. In such cases, the objective function should somehow favor those decompositions that reached an intermediate state that is "the closest" possible from the goal, for some metric relative to the difficulty for the embedded planner to solve the corresponding planning problem. As already discussed, no such distance exists, and the syntactic distance was used. Note also that both the distance to the goal (to be minimized), or the distance from the initial state (to be maximized) can thus be estimated.

3 CPT and subplan compression

CPT is a constraint-based optimal temporal planner and belongs to the POCL family [14]. It is available from the author's web site. This Section discusses the reasons for choosing CPT as the planner embedded inside D&E.

3.1 An optimal planner

The first motivation for choosing CPT is that it is an optimal planner, and a rather efficient one according to the results from the last IPCs. Although optimality is not mandatory, as *Divide-and-Evolve* anyway is bound to compute suboptimal plans, having the best possible plans joining two intermediate states is clearly an advantage. Indeed, if global solutions of good quality could not be obtained with an optimal planner, they would have probably been even harder to get with a suboptimal planner.

Furthermore, when CPT is used by *Divide-and-Evolve* to find subplans for each intermediate subproblem, it can provide several useful information for *Divide-and-Evolve*: the partial plan in the form described below, the makespan and number of backtracks that have been used for computing the fitness, and the complete state resulting from the application of the subplan to the initial state of the subproblem. Indeed, intermediate states that define a decomposition are likely to be incomplete states, as discussed in Section *Search Space* above. Hence they have to be completed in order to be valid initial states for the next subproblem.

Moreover, CPT also allows the user to specify a *maximum number of backtracks* to be used to solve a given problem. This gives a lever to a priori bound the difficulty of all subproblems CPT will be asked to solve, as discussed in Section *Search Space* above. Experimental results will demonstrate that the results of the *Divide-and-Evolve* approach are very sensitive to this parameter, that will be further discuss in the *Discussion* Section.

Another reason for using CPT is that it provides an easy access to some helpful data used by the EA for building individuals: possible atoms that can be used to select candidates for decomposition, and mutual exclusion relations between conflicting actions (mutex) in order to build mutex-free intermediate subproblems. The mutex computed by CPT are as powerful as Graphplan's mutexes, although computed in a different way. They nevertheless allow to check pairwise exclusions, avoiding many inconsistent intermediate states during the initialization and variation steps.

3.2 Subplan compression

But the main reason motivating the choice of CPT is probably its internal representation of partial plans, which allowed the design of an efficient compression routine based on causal links and precedence constraints. As usual in POCL planning, a partial plan is represented in CPT by a tuple $\langle Steps, Ord, CL, Open\rangle$ where *Steps* is the set of actions in the partial plan, *Ord* is a set of precedence constraints on *Steps* of the form $a_1 \prec a_2$, CL is a set of causal links of the form $a_1[p]a_2$ where $a_1, a_2 \in Steps$ and $p \in add(a_1) \cap pre(a_2)$, and *Open* is a set of open conditions (preconditions or goals not supported yet by a causal link of CL). *Steps* also contains two dummy actions *Start* and *End* with zero

durations, the first with an empty precondition and effect the initial state of the problem; the latter with precondition the goal of the problem and empty effects. These actions are used to initialize the first partial plan given to the planner, that will be refined up to a solution. A solution plan is a partial plan with no open condition and no flaws, which can be either causal links threatened by actions in the plan that must be protected by precedence relations, or unordered effect-conflicting actions.

Let $P = \langle A, O, I, G\rangle$ be a temporal planning problem and $(s_i)_{i\in[0,n+1]}$ be a decomposition of P, given by the EA, that admits a solution. We have then $s_0 = I$, $s_{n+1} = G$ and the dummy actions $Start$ and End such that $add(Start) = I$ and $pre(End) = G$. To each partial state s_i with $i \in [1, n+1]$, is associated a partial plan (computed by CPT) $\sigma_i = \langle Steps_i, Ord_i, CL_i, Open_i\rangle$ with dummy actions $Start_i$ and End_i such that $add(Start_i)$ is the complete state obtained by the successive application of the subplans $\sigma_1, \ldots, \sigma_{i-1}$ starting from I, and $pre(End_i) = s_i$. As the decomposition admits a solution, all these partial plans are solution plans of their respective subproblems and are such that $Open_i = \emptyset$. The compression routine simply consists in feeding CPT with an initial partial plan $\sigma = \langle Steps, Ord, CL, Open\rangle$ such that:

- $Steps = \bigcup_{i\in[1,n+1]}(Steps_i \setminus \{Start_i, End_i\}) \cup \{Start, End\}$;
- $Ord = \bigcup_{i\in[1,n+1]}\{a_1 \prec a_2 \in Ord_i \mid a_1 \neq Start_i \wedge a_2 \neq End_i\}$;
- $CL = \bigcup_{i\in[1,n+1]}\{a_1[p]a_2 \in CL_i \mid a_1 \neq Start_i \wedge a_2 \neq End_i\}$;
- $Open = \bigcup_{i\in[1,n+1]}\{p \mid Start_i[p]a \in CL_i\} \cup G$.

The problem solved by CPT is then $P' = \langle\{f \in A \mid \exists a \in Steps, f \in pre(a) \cup add(a) \cup del(a)\}, Steps, I, G\rangle$, starting from the initial partial plan σ. Furthermore, CPT is run in its 'canonical' mode[1], making the compression problem close to a scheduling problem: all the actions of $Steps$ already belong to the partial plan, no new action can be used, so only a valid schedule respecting at least the constraints of Ord and CL has to be found. Optimally solving this problem is NP-complete, as in the worst case, all subplans only contain one action: no precedence relation or causal link can then be used. In that case, all possible schedules between actions have then to be considered. However, this problem is not PSPACE-complete as the makespan of the global compressed plan is polynomially bounded by the sum of the subplan makespans, as their simple concatenation following the sequence of the individual computed by the EA is a (suboptimal) solution of P'.

In summary, the compression step considers all possible information of the solution subplans unrelated to intermediate initial and final states, and lets CPT look for the best possible way to connect these partial plans to each other and to the initial and goal states of the problem at hand. With respect to the information of the solution plans which is kept (actions, precedences, causal links), CPT finds the optimal compressed global plan. To obtain even better solutions, it is also possible to avoid considering some of these constraints, e.g. precedences, or causal links, or both. However, the choice to keep all this information was experimentally demonstrated to actually yield the best trade-off between speed and quality.

[1] The first version of CPT was solving the so-called canonical planning problem: each action in the problem can only be entered once into a partial plan.

4 Divide-and-Evolve

4.1 Choosing an Optimization Algorithm

As advocated in [12], the first ingredient for Planning Decomposition is a decomposition principle. Previous works have tackled this issue by relying on First Principles. On the opposite, *Divide-and-Evolve* uses an optimization algorithm without any knowledge of what a planning problem is to search the space of possible decompositions and optimize an objective function, as described in the previous Section. From those descriptions, however, the possible optimization algorithms are very few, as they must be able to search the unstructured space of variable length lists of (possibly incomplete) problem states, on which no metric is available, in order to optimize an objective function that is conditionally defined on different subsets of this search space. Evolutionary Algorithms (EAs) are general purpose optimization algorithms that have been demonstrated to be flexible and robust enough to handle such challenging optimization problems. In particular, several EA successes have been obtained in similar contexts of unstructured spaces, e.g. parse trees in the case of Genetic Programming for Analog Circuit Design [10].

4.2 Evolutionary Algorithms

Evolutionary Algorithms [5] are metaheuristic search methods based on a metaphor of the Darwinian evolution of biological populations, where candidate solutions of the optimization problem at hand are viewed as *individuals*. The emergence of adapted individuals (i.e. good solutions) results from the synergy between two phenomena: *natural selection* (the fittest individuals, with respect to the environment, survive and reproduce) and *blind variation* (the genetic material is randomly modified when passed on from the parents to their offspring during reproduction). The selection stage biases the choices of the algorithm towards candidate solutions with good values of the objective function (also termed *fitness*), whereas blind variation operators foster exploration of the search space by creating new individuals at random.

Among variation operators, one generally distinguishes crossover, where several individuals are recombined to give one offspring, and mutation, where a single individual is randomly modified to generate an offspring.

Because selection procedures are problem independent, implementing an evolutionary algorithm for a new problem only requires to define the search space (or, equivalently, the *representation* of candidate solutions), the fitness function, the variation operators and a generation procedure for the initial population.

However, after designing the above-mentioned components of an EA for a given problem, many parameters remain to be tuned in order to get the best out of the algorithm (e.g. population size, selection pressure, variation operator probabilities, ...). As it is the case for all search procedures, tuning an EA requires to solve the intensification-diversification dilemma (termed in the EA community *exploitation-exploration*): at any given stage of the algorithm, one should decide either to look around the best solutions obtained so far, hoping that some even better solutions lie there, or to look into yet unexplored regions of the search space where much better solutions might exist. Increasing the selection pressure, for instance, favors exploitation (intensification) while

increasing the probability (or the "strength") of mutation increases the exploration (diversification). Unfortunately, the theory of EAs provides today no guidance for parameter tuning. Moreover, because EAs are stochastic algorithms, their performance can only be validated statistically, from the results of multiple runs. Statistical approaches are used both to tune the algorithm parameters (derived from standard Design Of Experiments procedures) and to assess the results when comparing different settings (using standard statistical tests).

To the best of our knowledge, there have been very few attempts in the past to apply Evolutionary Algorithms to planning problems, and most works use a direct encoding of partial plans, such as [2].

4.3 Representation

As described in previous Section, individuals are represented as variable length lists of states. Since the *Divide-and-Evolve* strategy was originally designed as a metaphor for some railway trajectory planning problem, intermediate states are called *stations*. Note that states s_0 and s_{n+1}, being respectively the initial state and the goal of the problem at hand, will not be modified by evolution, and hence are encoded in the individuals.

In the present work, stations only contain atoms built on predicates of arity 1 or 2. Mutual exclusion can easily be checked with the help of CPT data structure, that is grounded at the beginning of each run, and can be asked for any pair of atoms. Atom selection for station generation, at the initialization stage or within variation operators, is predicate oriented. A first simple choice is to use goal predicates, as successfully demonstrated in preliminary previous work [not cited]. However there is no reason to restrict station generation to goal predicates. Consider for instance the `zeno 13` IPC-3 benchmark in which only the `AT` predicate appears in the goal description. Experimental results have shown that an optimal solution can be obtained using either one, or two, or all predicates to represent the stations. The two following optimal decompositions illustrate this fact: they were obtained using different predicates, and resulted in very different plans, though both optimal. The subsequent Table shows the corresponding behaviors of D&E (number of backtracks (Btk.) and makespan (Mksp.) obtained by CPT on subproblems, and makespan after compression).

4.3.1 Using predicate IN

$$s_1 = \{(\text{IN person2 plane2})\}$$

$$s_2 = \{(\text{IN person8 plane3}),(\text{IN person4 plane1})\}$$

4.3.2 Using predicate AT

$$s_1 = \{(\text{AT plane1 city5}),(\text{AT person5 city2}),(\text{AT person4 city5})\}$$

	IN		**AT**	
Subproblem	Btk.	Mksp.	Btk.	Mksp.
$Init \rightarrow s_1$	0	120	0	250
$s_1 \rightarrow s_2$	0	120	10	380
$s_2 \rightarrow Goal$	10	496		
$\sum$	10	736	10	630
Compression	0	596	0	596

As of today, the choice of what predicates should be selected for intermediate state generation remains an open question, and will be investigated in further research. The size of the search space is of course related to the number of predicates, and discarding some predicates might forbid trajectories to optimal solutions. The usual trade-off has to be found between the size of the search space and the expressivity of the representation.

4.4 Fitness

As discussed in Section *Objective Function*, the computation of the fitness starts by running CPT on all subproblems sequentially, stopping when CPT fails to solve one. If all subproblems have been solved, CPT is used again for the compression of all subplans, as described in Section *CPT*.

The fitness (to be minimized) is finally computed using one of two different formulas, depending on whether a failure has occurred or not:

$$Fitness = \begin{cases} m + \frac{n-u+1}{m} & \text{all subproblems solved} \\ \\ (n-u+1) & \text{failure on one subproblem} \end{cases} \tag{1}$$

where m is the total makespan of the compressed plan, n the number of stations of the individual, u is the number of *useful* stations in the individual. A station is considered useful if the plan to reach it from previous station has a not null makespan. The idea behind the $(n-u+1)$ term in case of failure is to promote those individuals that contain the largest number of useful stations.

Note that these formulas are used to compare two feasible individuals (first formula) or two unfeasible individuals (second formula). However, a feasible individual is always preferred to an unfeasible one.

4.5 Variation operators

The variation operators modify the individuals in order to explore the search space, i.e. the space of lists of intermediate states. However, it is highly desirable that all variation operators build as few inconsistent states as possible, and using CPT mutex information was an easy first step in that direction. Moreover, these operators should ensure the *ergodicity* of the resulting stochastic process: any point of the search space must be reachable from any other point using a finite number of variation operators with non-zero probability. Two types of variation operators are recognized: *mutations* generate offspring from a single parent, while *crossovers* use two or more parents. They will be described in turn.

4.5.1 Crossover operator

The crossover operator used in this work only considers the station level, i.e. only exchange stations. The basic 1-point crossover is used here: Assuming the recombination of two individuals $(s_i)_{1\leq n}$ and $(t_i)_{1\leq m}$, the 1-point crossover amounts to uniformly choosing one station in each parent, say s_a and t_b, and exchanging the second parts of both lists of stations. This leads to the two offspring $(s_1, ..., s_a, t_{b+1}, ..., t_m)$ and $(t_1, ..., t_b, s_{a+1}, ..., s_n)$. Of course, the length of the offspring is likely to differ from those of the parents.

4.5.2 Mutations

A mutation operator can act here at two levels: at the individual level, considering each stations as one gene; or at the station level, modifying some atoms in a given station. Because an individual is a variable length list of stations, and a station is a variable length list of atoms, some mutation operators will simply add or remove a station, or an atom in a station. More precisely, several mutation operators have been designed, to address the issues raised above. They will now be described in turn, assuming parent $(s_1, \ldots, s_{last}, \ldots, s_n)$, where s_{last} is the last station that CPT reached when computing the fitness of this individual (with $last = n + 1$ if all subproblems have been solved by CPT).

The **addStation** mutation uniformly chooses $i \leq min(n, last)$. A new station S_{new} is built, containing all common atoms of s_i and s_{i+1}, some random atoms belonging to either s_i or s_{i+1}, and random atoms created from scratch. Every time an atom is chosen, it is added to s_{new} only if it is not *mutex* with the existing atoms. Station s_{new} is then inserted between stations s_i and s_{i+1}. The idea of this procedure is to insert a station that is close to actually being "between" two existing stations of the parent, at least syntactically speaking.

The **delStation** mutation removes station s_i from the parent, where i is uniformly chosen in $[1, min(n, last + 1)]$.

The **changeAtom** mutation modifies one atom in a station s_i where i is uniformly chosen in $[1, min(n, last + 1)]$. The modification is done by first randomly choosing one atom of station s_i, and looking for possible atoms using the same predicate (and, in case of predicates of arity 2, the same first argument) that is pairwise consistent with other atoms of station s_i (i.e. not mutex).

Finally, the **delAtom** mutation randomly removes one atom from station s_i, where i is uniformly chosen in $[1, min(n, last + 1)]$.

4.5.3 Applying variation operators

Several parameters control the application of the variation operators. During an evolutionary run, two parents are chosen according to the selection procedure at hand. With probability p_{cross}, they are recombined using the crossover operator. Each one then undergoes mutation with probability p_{mut}. When an individual must undergo mutation, 4 additional user-defined relative *weights* ($w_{addStation}$, $w_{delStation}$, $w_{changeAtom}$, $w_{delAtom}$) are used to choose among the 4 mutation operators defined above: each operator has a probability proportional to its weight of being applied. At most one mutation operator is thus applied to each individual.

4.6 Initialization of the population

Two approaches have been tried. The **blind initialization**, as its name indicates, simply generates atoms from the chosen predicates. The number of stations is first uniformly chosen within some user-defined bounds. The number of atoms to be constructed is then uniformly chosen within some user-defined bounds. Atoms are then sequentially added until the chosen number of atoms; atoms that are mutex with one of the already existing atoms are rejected.

The **goal-directed initialization** also starts by choosing randomly the number of stations within some user-defined bounds. But it only considers those atoms that are present in the goal of the problem at hand. This procedure can be seen as randomly spreading the atoms from the goal over the chosen number of stations.

5 Experimental results

This Section presents experimental results illustrating the efficiency of D&E on several simple-time domains that were used for the 3rd and 4th IPC.

D&E was implemented within the *Evolving Objects* library (http://eodev.sourceforge.net), a template-based, ANSI-C++ compliant Evolutionary Computation Open Source library. Experiments were done using a Intel Xeon CPU 3.60 GHz computer with 4 Gb of RAM, running Linux.

5.1 The domains

Results on only three domains are presented here, all three from IPC benchmarks: `zeno` and `rover` from IPC-3, and `satellite` from IPC-4. Only instances large enough to be of some interest are reported. On some other benchmarks, the results of D&E are very similar in quality to those obtained on one of the three presented cases, but will not be presented here for space reasons: on `blocks` (similar to `zeno`), on `drivers` and `depots` (similar to `satellite`).

Due to some limitations of CPT 3.0 with respect to temporal semantics, some benchmark domains (such as `depots`, `drivers`, and `satellite` from IPC-3) are not presented here, as CPT alone obtains worse results than LPG-*TD*. No doubt that further versions of CPT will address this issue, and *Divide-and-Evolve* will automatically benefit from those improvements.

However, on the `pipesworld` benchmarks, the current *Divide-and-Evolve* approach failed to give any interesting result: this still needs to be investigated in depth. And finally, the recent benchmarks from IPC-5 were not yet tested, by lack of time.

5.2 The Reference Planners

The results of D&E have been compared to two well-known temporal planners.

SGPlan 5 is based on subgoal partitioning, and seemed a good competitor for D&E. However, the source code was not made available by the authors, and the only available results are those from the cited paper. They are hence now rather old, and should probably be updated to reflect the state-of-the-art of SGPlan: they were constantly outperformed by both D&E and LPG-*TD* and will not be reported here.

LPG-*TD* is based on a stochastic local search in the space of temporal action graphs [6]. It was downloaded from the authors'web site, and all reported results have been run locally, using local search, no best-first search, and a maximum number of desired solutions of 100. Those runs were done on the same computers than D&E, allowing a one month time limit.

5.3 *Divide-and-Evolve* settings

Only the predicates that are present in the goal of the problem at hand were used to represent stations.

A difficult (and often underestimated) part of evolutionary successes is the setting of the numerous parameters of the algorithm. Indeed, there exists no theoretical guidelines even to tune the standard probabilities of crossover and mutation (p_{cross} and p_{mut}). Users generally rely on their previous experiences in Evolutionary Computation, or use standard statistical methods, e.g. Design Of Experiments (DOE) and ANalysis Of VAriance (ANOVA). In any case, setting the parameters of an EA applied to a brand new domain such as Evolutionary Planning Decomposition is a tedious task.

Because there are too many parameters to tune here, some of them were set once and forall after the preliminary experiments reported in the authors' previous work [not cited]. The evolution engine has been chosen to be a (10+70)-ES: 10 parents generate 70 offspring (no selection at this point), and the best of those 80 individuals become the parents of the next generation.

Also, the number of stations in an individual during initialization is uniformly chosen between 1 and the number of atoms in the goal of the problem; the number of atoms per station is chosen in $[1, 4]$; and the initialization is goal-directed. Last but not least, the number of backtracks that CPT is allowed to use for solving each subproblem is arbitrarily set to 4000 after intensive tests on the small `zeno` instances (≤ 13).

The remaining parameters regard the variation operators: the probabilities of individual-level application of crossover and mutation (p_{cross} and p_{mut}) and the relative weights of the 4 mutation operators ($w_{addStation}$, $w_{delStation}$, $w_{changeAtom}$, $w_{delAtom}$). A two-stage DOE was used: first, the relative weights were set to (4, 1, 4, 1) (from preliminary experiments), and an incomplete factorial DOE was done on the parameters (p_{cross}, p_{mut}) using `zeno 10-12` problems with 11 runs per parameter set. The differences in mean were then validated using both Kolmogorov and Wilcoxon non-parametric tests at 95% confidence level. Three pairs for p_{cross} and p_{mut} were found significantly better than the others, and another DOE on the 4 weights and those pairs yielded the final setting: (0.25,0.75) for (p_{cross}, p_{mut}), and (35, 3, 35, 7) for the relative mutation weights.

5.4 Results

Table 1 displays a summary of the results on the three cited domains for CPT alone, D&E and LPG.

5.4.1 Validating the *Divide-and-Evolve* approach

The first clear conclusion that can be drawn from those tables is that indeed, D&E is able to solve large instances that CPT cannot solve, even when given one month CPU time. This is true for all large instances of all 3 domains, even if sometimes with very small success rate. For some instances, "$\leq$ *value*" for CPT indicates that, given this bound, CPT was able to find a solution, but it was not able to prove its optimality. Such bounds were obtained either from LPG-TD or from D&E.

Note that on some `zeno` small instances, that CPT alone can solve, D&E solutions were decompositions using all atoms of the goal in 1 or 2 stations, such solutions being found in initial population, thanks to the goal-based initialization. However, the *Divide-and-Evolve* approach is clearly doing what it was designed for: finding decompositions into simpler subproblems – simpler here at least for CPT.

5.4.2 D&E versus LPG

Two different situations can be seen on the Table: on `zeno` domain, LPG clearly outperforms D&E on large instances, in term of solution quality – though sometimes at a very high computational cost (e.g. 11 and 13 days vs. approximately 2 days on instances 16 and 19).

The situation is very different on `satellite` domain, where D&E clearly outperforms LPG in term of solution quality for all large instances (except `satellite 24`). Comparing the computational costs of D&E and LPG, a clear pattern appears if also looking at the performance of CPT alone: for small instances (≤ 19), that CPT alone can solve, D&E requires very little time, even though it cannot find the obvious solutions (spreading the atoms of the goal across a few stations) due to the limitation in the number of backtracks. But on larger instances (≥ 20), LPG stops after minutes while D&E requires days, but finds better solutions.

The situation is somewhat intermediate on the `rover` domain, where both algorithms find exactly the same best solution for almost all instances (D&E is slightly better for instance 6 and slightly worse for the two largest instances), but LPG requires again much less computational effort (seconds against minutes), except for instance 6, even though CPT alone cannot solve it.

6 Discussion

6.0.3 Complexity

Performance analysis in terms of number of backtracks can be a good indicator, but subject to discussion due to the highly erratic behavior of CPT in this respect. However this large variability in performance is not specific to CPT and [7] have observed similar runtime distributions of backtrack procedures, and shown how random restarts can dramatically improve the performance. For instance, on `zeno 13`, CPT needs only 3 backtracks to find a solution at bound 596 (whereas D&E needs 10 backtracks), but uses 36674 backtracks to prove there is no solution at bound 595. On `rover 5` on the other hand, CPT used 16873 backtracks at the optimal bound, whereas D&E found a solution

with 3 stations which consumed 9 backtracks only, including compression. In general however, because the search for each subproblem is artificially limited in terms of number of backtracks, D&E needed much less backtracks to solve the optimal decomposition, compared to the amount needed by CPT alone. In this sense, the number of backtracks can be seen as a good a posteriori indicator of complexity.

6.0.4 Embedding Other Planners

It has been already argued that, though any planner could be used in lieu of CPT to solve the subproblems of each decomposition, the critical issue is the compression step, easily done at the moment by CPT. Choosing another planner (e.g. LPG-*TD*) would require to rewrite a compression module.

Hence, a possible suboptimal planner could be eCPT [13], a suboptimal version of CPT that attempts to solve temporal planning problems with the same framework (constraint-based, POCL planning) in a backtrack-free way. Indeed, very preliminary trials on `zeno 11-14` demonstrated that D&E using suboptimal CPT could solve the global problem to optimality using CPU time one order of magnitude less than when using the optimal version of CPT. Further work will investigate this line of research.

However, one limitation of graphplan oriented planners is space consumption. Factored Planning, applying a *Divide-and-Conquer* strategy at the domain level [1, 9], offers another promising research direction.

6.0.5 Parameter tuning

It is well-known that parameter tuning is one of the weaknesses of EAs in general. D&E is not an exception, and though the parameters used in all experiments presented here seem to be rather robust, new domains will probably require new settings. Statistical techniques known as *Racing* seem to be a good approach to limit the number of experiments in the DOE for EAs [15]. On-going experiments already demonstrated that one of the most influential parameter is the limit on the number of backtracks set on CPT when solving the subproblems. Our hope is to be able to learn from extensive experimental data some empirical formula giving an estimation of the best value for this parameter from descriptive features of the instance at hand.

7 Conclusion

This paper has presented *Divide-and-Evolve*, an original approach to Planning Decomposition, based on an Evolutionary Algorithm searching the space of sequences of partial states. The optimal planner CPT is used to solve the different subproblems (reaching all intermediate states one after the other), but also to compress the concatenation of subplans so obtained, in order to take advantage of the possible concurrency. The results demonstrated that indeed D&E is able to solve problems that CPT cannot solve in months, and moreover can obtain on some problem instances better solutions than LPG-*TD*, another well-known suboptimal planner. A further benefit of *Divide-and-Evolve* approach, demonstrated in previous work [uncited here], is that it opens up the

road to multi-objective planning, by simply replacing the evolutionary engine used here by any standard Multi-Objective EA.

Many issues remain open, though, from the sound choice of the predicates to be used to represent the intermediate states to the automatic tuning of the many parameters D&E has at the moment. The choice of other embedded planner is another possible way to go, though the chosen planner must be able to address compression issue. More generally, deep theoretical investigations are needed to understand which domains are amenable to *Divide-and-Evolve*.

But the main drawback of D&E is its high computational cost and sometimes low reliability. However, because coarse-grain parallelization of EAs is straightforward, we believe that the original D&E approach to planning decomposition will help pushing further the limits of many existing temporal planners, offering them an efficient way to fully benefit from the modern multi-core computer architectures.

Table 1: Results from 50 independent runs: Best makespan obtained by all planners (first 3 lines), mean runtime for D&E and LPG-*TD*, and proportion of D&E runs that did find the best value in line 2. The values in bold are the best values obtained by D&E, and LPG. On CPT line, an asterisk indicates that the optimal makespan was not reached by any of the suboptimal planners. A dash indicates that no solution could ever be found during those experiments.

Zeno		11	12	13	14	15	16	17	18	19	20
3***Makespan**	CPT	423	549	596	476	$\leq$ 695	$\leq$ 626	$\leq$ 1377	-	$\leq$ 1444	-
	D&E	**423**	**549**	**596**	**476**	**702**	696	1437	1298	1673	-
	LPG	**423**	**549**	**596**	519	709	**626**	**1048**	**1062**	**1208**	**1508**
2***Time**	D&E	3min	27min	1h	3h	6h	30h	23h	2days	2days	-
	LPG	< 1s	3min	9h	16min	5h	11days	1day	10h	13days	1day
D&E% of success		100 %	39 %	89 %	11 %	2 %	28 %	16 %	6 %	20 %	-

Rover		6	8	9	10	11	12	13	14	15	16	17	18	19	20
3***Makespan**	CPT	$\leq$ 125	$\leq$105	$\leq$103	-	$\leq$105	88	$\leq$143	$\leq$107	$\leq$122	-	$\leq$168	$\leq$140	-	-
	D&E	**125**	**105**	**103**	**133**	**105**	**88**	**143**	**107**	**122**	**138**	**168**	**140**	228	250
	LPG	128	**105**	**103**	**133**	**105**	**88**	**143**	**107**	**122**	**138**	**168**	**140**	**223**	**248**
2***Time**	D&E	15min	12min	13min	13min	4h	2min	21min	12min	31min	6h	33min	4min	18h	1days
	LPG	3h	3s	53s	11s	89s	< 1s	5min	32s	24s	2min	3min	6s	5h	14min
D&E% of success		8 %	82 %	90 %	14 %	96 %	100 %	10 %	100 %	42 %	10%	19%	4 %	2 %	2 %

Satellite		11	12	13	14	15	16	17	18	19	20	21	22	23	24
3***Makespan**	CPT	115.84	129.75*	113.15*	85.77*	80.07*	77.71	74.89	66.46	99.59*	$\leq$150.2	-	-	-	-
	D&E	**115.84**	**130.46**	**115.46**	**86.13**	**80.77**	**77.71**	**74.89**	**66.46**	**101.02**	**148.66**	**126.70**	**201.05**	**186.14**	-
	LPG	125.73	156.18	170.60	106.90	113.93	90.35	87.55	92.27	140.58	206.82	245.58	293.58	385.91	501
2***Time**	D&E	91s	1h	2h	42min	20min	48min	49min	5min	1h	2h	1day	2days	7days	-
	LPG	12h	2h	16min	1h	6h	4h	1day	3h	17h	10min	24min	2days	14min	8min
D&E% of success		52 %	2 %	2 %	2 %	2 %	8 %	13 %	2 %	2 %	2 %	10 %	10 %	10 %	-

References

[1] R. I. Brafman and C. Domshlak. Factored Planning: How, When, and When Not. In *Proceedings of AAAI-06*, 2006.

[2] A. H. Brié and P. Morignot. Genetic Planning Using Variable Length Chromosomes. In *Proceedings of ICAPS-05*, 2005.

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

[4] Y. Chen, C. Hsu, and B. Wah. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *Artificial Intelligence*, 26:323–369, 2006.

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

[6] A. Gerevini, A. Saetti, and I. Serina. Planning through Stochastic Local Search and Temporal Action Graphs in LPG. *JAIR*, 20:239–290, 2003.

[7] C. Gomes, B. Selman, and N. Crato. Heavy-Tailed Distributions in Combinatorial Search. In *Proc. CP-97*, pages 121–135, 1997.

[8] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered Landmarks in Planning. *JAIR*, 22:215–278, 2004.

[9] E. Kelareva, O. Buffet, J. Huang, and S. Thiébaux. Factored Planning Using Decomposition Trees. In *Proc. IJCAI-07*, 2007.

[10] J. R. Koza and al. *Genetic Programming III: Automatic Synthesis of Analog Circuits*. MIT Press, 1999.

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

[12] L. Sebastia, E. Onaindia, and E. Marza. Decomposition of Planning Problems. *AI Communications*, 19(1):49–81, 2006.

[13] V. Vidal and H. Geffner. Solving Simple Planning Problems with More Inference and No Search. In *Proceedings of CP-05*, pages 682–696, 2005.

[14] V. Vidal and H. Geffner. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence*, 170(3):298–335, 2006.

[15] B. Yuan and M. Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. LNCS 3242, pages 172–181. Springer Verlag, 2004.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803