Divide-And-Evolve Facing State-of-the-art Temporal Planners during the 6th International Planning Competition

Jacques Bibai, Marc Schoenauer, Pierre Savéant

Divide-And-Evolve Facing State-of-the-art Temporal Planners during the 6^{th} International Planning Competition

Jacques BIBAI[1 2], Marc SCHOENAUER[1], and Pierre SAVÉANT[2]

[1] Projet TAO, INRIA Saclay & LRI, Université Paris Sud, Orsay, France.
firstname.lastname@inria.fr
[2] Thales Research & Technology, Palaiseau, France.
firstname.lastname@thalesgroup.com

Abstract. *Divide-and-Evolve* (DAE) is the first evolutionary planner that has entered the biennial International Planning Competition (IPC). Though the overall results were disappointing, a detailed investigation demonstrates that in spite of a harsh time constraint imposed by the competition rules, DAE was able to obtain the best quality results in a number of instances. Moreover, those results can be further improved by removing the time constraint, and correcting a problem due to completely random individuals. Room for further improvements are also explored.

1 Introduction

An artificial intelligence planning problem is specified by the description of an initial state, a set of desired goals to reach and a set of possible actions. An action modifies the current state, and can be applied only if certain conditions in the current state are met. A solution to a planning problem is an ordered set of actions forming a valid plan, whose execution in the initial state transforms it into a state where the problem goals are satisfied. In temporal planning problems a given goal is reached by taking a number of durative actions which may temporally overlap.

Although researchers have investigated a variety of methods for solving the temporal planning problems submitted to the biennial International Planning Competition (IPC) since 1998 (e.g. extended planning graph: TGP [13], LPG [7, 6]; reduction to linear programming: LPGP [8]; reduction to constraint programming: CPT [14, 15]; partitioning planning problems into subproblems by parallel decomposition: SGPlan [3]), none of them rely on an evolutionary algorithm. Recently, *Divide-and-Evolve* (DAE), an Evolutionary Planner was proposed by the authors [11, 12], and entered the IPC-6 competition, becoming, to the best of our knowledge, the first evolutionary temporal planning system that participated to such competition.

DAE hybridises evolutionary algorithms with classical planning methods by dividing the initial problem into a sequence of subproblems, solving each subproblem in turn, and building a global solution from the subproblem solutions.

This approach is both original and generic for the planning community. However, the results of DAE in the IPC-6 competition were altogether disappointing, even though it performed best on several problems.

In this paper, we investigate the reasons for such results, and propose several ways to enhance DAE performances. Section 2 briefly introduces Temporal Planning Problems (TPP); Section 3 details the *Divide-and-Evolve* approach, representation, fitness function and variation operators; section 4 validates DAE beyond [11, 12] by presenting experimental results on previous IPC benchmarks; Section 5 details the results of DAE during the IPC-6 competition, then proposes some enhancements that are validated on the IPC-6 benchmarks.

2 Temporal Planning Problem

Domain-independent planners rely on the Planning Domain Definition Language (PDDL) [10], inherited from the STRIPS model [4], to standardise and represent a planning problem. It was developed mainly to make the International Planning Competition (IPC) series possible. The planning competition compares the performance of candidate planners on a set of benchmark problems, thus requiring a common language for specifying them. The language has been extended for representing temporality and action concurrency in PDDL2.1 [5].

The description of a planning problem splits in two separate parts: the generic domain theory on one hand and a specific instance scenario on the other hand. The domain definition specifies object types, predicates and actions which capture the possible moves, whereas the instance scenario declares the objects of interest, the initial state and the goal description. A state is described by a set of atomic formulae, or atoms. An atom is defined by a predicate symbol from the domain followed by a list of object identifiers: *(PREDICATE_NAME* OBJ_1 *...* OBJ_N*)*. The initial state is complete, i.e. it gives a unique status of the world, whereas the goal is only partial and can be true in many different states. An action is composed of a set of preconditions and a set of effects, and applies to a list of variables given as arguments. Preconditions are logical constraints which apply domain predicates to the arguments an trigger the effects when they are satisfied. Effects enable state translations by adding or removing atoms.

A solution to a temporal planning problem is a consistent schedule of grounded actions whose execution in the initial state leads to a state where the problem goal is satisfied. The total duration of the solution plan is called the *makespan* and its minimisation is the usual objective of the optimisation.

A planning problem defined on domain D with initial state I and goal G will be denoted $\mathcal{P}_D(I, G)$ in the following.

3 Divide-and-Evolve

Divide-and-Evolve (DAE) is an evolutionnary computation technique that searches the space of state decompositions [11, 12]. In order to solve a planning problem $\mathcal{P}_D(I, G)$, the basic idea is to find a sequence of states $S_1, \ldots, S_n$, and to use

some 'local' planner to solve the series of planning problems $\mathcal{P}_D(S_k, S_{k+1})$, for $k \in [0, n]$ (with the convention that $S_0 = I$ and $S_{n+1} = G$). The concatenation of the plans solving all subproblems is then a plan that solves the original global problem. The rationale is that all sub-problems $\mathcal{P}_D(S_k, S_{k+1})$ can be made simpler for the local planner than the original problem $\mathcal{P}_D(I, G)$, thus allowing DAE to find solutions of instances that the local planner alone cannot solve. Very preliminary validation on the IPC-3 `zeno` benchmarks validated this idea in [11, 12]. However, many improvements have been brought to DAE since those early results, and, more importantly, comparison of DAE results with state-of-the-art planners remained to be done.

We shall now present DAE, specifically detailing the problem-specific representation and associated initialisation and variation operators, as well as the fitness function.

3.1 Representation

Following the rationale above, an individual in DAE is a sequence of states. And as described in Section 2, a state is a list of boolean atoms. However, searching the space of complete states would result in a very fast combinatorial explosion of the search space size. Moreover, goals of TPPs need only to be defined as partial states. It thus seemed practical to search only sequences of partial states. However, this raises the issue of the **choice of the atoms** to be used to represent individuals, among all possible atoms. The choice made in the first versions of DAE [11, 12] was to use only the predicates that appear in the goal – this variant of DAE will be termed DAE1 in the following. Another option is to use the results of the grounding step of CPT, that generates all possible atoms, and to choose the predicates (at most 3) that are more frequent. This variant is termed DAE2 in the following.

Nevertheless, even when restricted to specific choices of atoms, the choice of random atoms can lead to inconsistent partial states, because some sets of atoms can be *mutually exclusive* (`mutex` in short). Whereas it could be possible to allow `mutex` atoms in the partial states generated by DAE, and to let evolution discard them, it seems more efficient to a priori forbid them, as much as possible. Because the embedded planner CPT computes and maintains a list of all `mutex` pairs of atoms, it is thus possible to exclude such pairs a priori, even though this still does not guarantee that the partial state is consistent – but determining if a state is consistent amounts to solving the complete planning problem!

An individual in DAE is hence represented as a variable length ordered list of partial states, and each state is a variable length list of atoms involving only predicates that are present in the goal G that are not pairwise `mutex`.

3.2 Fitness, and CPT

The fitness of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling a local planner to solve the sequence of problems $\mathcal{P}_D(S_k, S_{k+1})$ $(k = 0, \ldots, n)$. Any existing planner could be used here, and DAE uses CPT, an exact planning

system for temporal STRIPS planning. CPT combines a branching scheme based on Partial Order Causal Link (POCL) Planning with powerful and sound pruning rules implemented as constraints [14, 15].

For any given k, if CPT succeeds in solving $\mathcal{P}_D(S_k, S_{k+1})$, the final complete state is computed, and becomes the initial state of next problem: initial states need to be complete (and denoting each problem as $\mathcal{P}_D(S_k, S_{k+1})$ is indeed an abusive notation). If all problems, $\mathcal{P}_D(S_k, S_{k+1})$ are solved by CPT, the individual is called *feasible*, and the concatenation of all solutions plans for all $\mathcal{P}_D(S_k, S_{k+1})$ is a global solution plan for $\mathcal{P}_D(S_0 = I, S_{n+1} = G)$. However, this plan can in general be optimised by parallelising some of its actions, in a step call *compression* (see [12, 2] for detailed discussion). The fitness of a feasible individual is the makespan of the compressed plan.

However, as soon as CPT fails to solve one $\mathcal{P}_D(S_k, S_{k+1})$ problem, the following problem $\mathcal{P}_D(S_{k+1}, S_{k+2})$ cannot be even tackled by CPT, as its complete initial state is in fact unknown, and no makespan can be given to that individual. All such plans receive a fixed penalty cost such that the fitness of any infeasible individual is higher than that of any feasible individual. In order to nevertheless give some selection pressure toward feasible individuals, the relative rank of the first problem that CPT fails to solve is added to the fixed penalty, so infeasible individuals which solve the more subproblems are favoured by selection.

Finally, because the initial population contains randomly generated individuals, some of them might contain some subproblems that are in fact more difficult than the original global problems. Because CPT can sometimes take months to solve very difficult problems, it was necessary to limit the **maximal number of backtracks** that CPT is allowed to use to solve any of the subproblems. And because, ultimately, it is hoped that all subproblems will be easy to solve, such limitation should not harm the search for solutions – though setting this limit might prove difficult (see Section 5.1).

3.3 Initialisation and Variation Operators

The **initialisation** of an individual is the following: First, the number of states is uniformly drawn between one and the number of atoms in the goal of the problem, divided by the number of atoms per state; the number of atoms per state is chosen uniformly in $[1, 4]$. Atoms are then chosen one by one, uniformly in the allowed set of atoms (i.e. built on the goal predicate in most results here), and added to the individual if not `mutex` with any other atom already there (thanks to CPT list of pairwise exclusions).

A 1-point **crossover** is used, adapted to variable-length representation in that both crossover points are uniformly independently chosen in both parents.

Four different mutation operators have been designed, and once an individual has been chosen for mutation (according to a population-level mutation rate), the choice of which mutation to apply is made according to user-defined relative weights (see Section 3.4). Two mutation operators act at the individual level, either removing or inserting a partial state at a uniformly chosen position in the list, and two act at the state level, removing or modifying an atom (while

still avoiding pairwise `mutex` atoms) in a uniformly chosen partial state of the individual. A mutation that adds an atom to an existing partial state was used in early experiments, but soon was found to have no influence whatsoever on the results, and hence was abandoned. The reason for that is probably that when a new partial-state is inserted in the individual, it is created using some atoms of its two neighbors, plus some new atoms, thus bringing in diversity at the state level.

3.4 Evolution Engine and Parameter Settings

One of the main weaknesses of Evolutionary Algorithms today is the difficulty in tuning their numerous parameters, for which there exist no theoretical guidelines. Users generally rely on their previous experience on similar problems, or use standard but expensive statistical methods, e.g. Design of Experiments (DOE) and Analysis of Variance (ANOVA).

However, because there are here many parameters to tune, some of them were set once and for all based on preliminary experiments [11, 12]. This is the case for the **evolution engine**, chosen to be a (10+70)-ES: 10 parents generate 70 offspring using variation operators, and the best of those 80 individuals become the parents of the next generation. The same stopping criterion has also been used for all experiments: after a minimum number of 10 generations, evolution is stopped if no improvement of the best fitness in the population is made during 20 generations, with a maximum of 100 generations altogether.

The remaining parameters concern 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 p_{cross} and p_{mut} on `zeno 10-12` problems (11 runs per parameter set). The differences were then validated using both Kolmogorov and Wilcoxon non-parametric tests at 95% confidence levels. Three pairs for (p_{cross}, p_{mut}) were found significantly better than the others, and another DOE on the 4 weights and those 3 pairs yielded the final setting: (0.25, 0.75) for (p_{cross}, p_{mut}), and (35, 3, 35, 7) for the relative mutation weights.

4 Comparing DAE and CPT on IPC-3 Problems

This section presents experimental results that further validate the DAE approach beyond the initial results in [11] using several other domains from the 3^{rd} International Planning Competition.

Significantly, DAE can solve several problems that CPT alone cannot. For instance, for the `zeno` domain (not shown on the figure 1), DAE solved the first 19 instances when CPT failed after instance 14. Same thing is true, though not as clearly related to the instance number, for the `satellite` and `drivers` domains, and to a lesser extent `depot`.

The other observation concerns the quality of the results, compared to the optimal values found by CPT, an exact planner. For all instances of the `rovers`, `zeno` and `drivers` domains, DAE has found optimal values. For the `satellite` and `depot` domains, DAE always found either the optimal value, or a best makespan very close to the optimum (more than 93%). Moreover, when CPT fails to find a solution, the values found by DAE are very close to (and sometimes better than) those found by LPG [2].

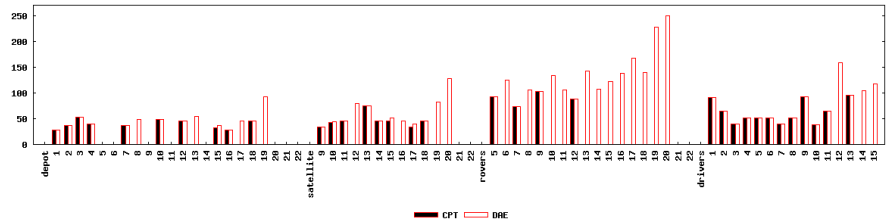

Fig. 1. Optimal makespans for CPT (1 month time limit) and DAE (best of 100 runs, 4000 backtracks limit) on `depot, satellite, rovers and drivers` IPC-3 domains. Each column represents an instance of the domain.

5 DAE at the IPC-6 Competition

The International Planning Competition (IPC) is a biennial event organised within the International Conference on Planning and Scheduling, aiming at analysing and advancing the state-of-the-art in automated planning systems. The experimental conditions imposed by this competition are very simple: all planners have to solve several instances of different planning domains in a completely automated way, and within 30min of CPU time.

5.1 Maximal number of backtracks

Divide-and-Evolve entered the 6^{th} edition in the deterministic track [1] with DAE1 and DAE2, the two versions described in Section 3.1 (DAE1 uses the predicates of the goal to represent the individuals, while DAE2 uses those having the most instances amongst all possible atoms).

Whereas all parameters described in Section 3.4 were chosen as default, giving a fixed value to the maximum number of backtracks allowed for CPT had two possible major drawbacks: on the one hand, a too small limit could completely prevent DAE from finding solutions even to problems that CPT alone could solve (using more backtracks); on the other hand, a too large limit would allow CPT to spend a lot of time on poor individuals in the early generations, slowing down

the evolution and forbidding any good solution to be found within the 30min limit.

Based on numerous experiments made on IPC-3 benchmarks, the maximum number of backtracks allowed for CPT was set to a linear combination of the ratio of the number of causal links plus the number of actions to the number of atoms generated by those actions, and the ratio of the number of nodes to the number of conflicts:

$$bks_{state} = \#Ga * (\frac{\#nodes}{\#conflicts} + 2 * (\frac{\#causals + \#actions}{\#atoms})) \quad (1)$$

where $\#Ga$ is the number of goal atoms, $\#causals$ the number of causal links, $\#actions$ the number of actions and $\#atoms$ the number of atoms generated after action grounding.

Further experiments have also demonstrated the need for more backtracks when solving the last subproblem (reaching the global goal). Hence a specific formula was designed for this case: $bks_{goal} = 7 * bks_{state}$

5.2 Detailed DAE Results at IPC-6

Using the above automated parameter settings, the raw global result of DAE are the 4^{th} and 5^{th} ranks among 6 participants. Those poor results can however be refined when considered domain by domain: In two of the 6 domains, CPT could not even complete its grounding step in less than 30min: in such domain, of course, there is no hope that DAE can solve any of the instances. Furthermore, though in the 4 other domains DAE could not solve all instances, it almost always found a better makespan than *SGPlan6*, the winner of the competition, on instances it did solve, as witnessed by Figures 2, 3, 4 and 5 for, respectively, the `peg solitaire`, the `openstack`, the `parcprinter` and the `crewplanning` domains.

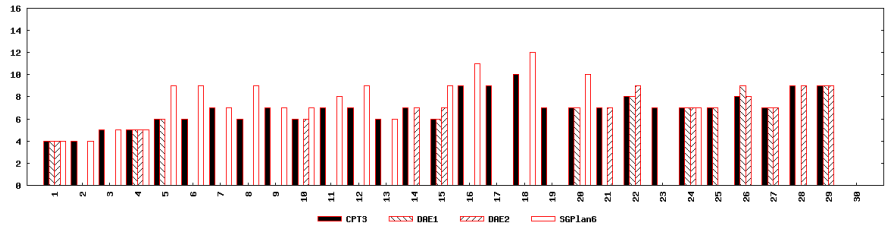

Fig. 2. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `peg solitaire` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance).

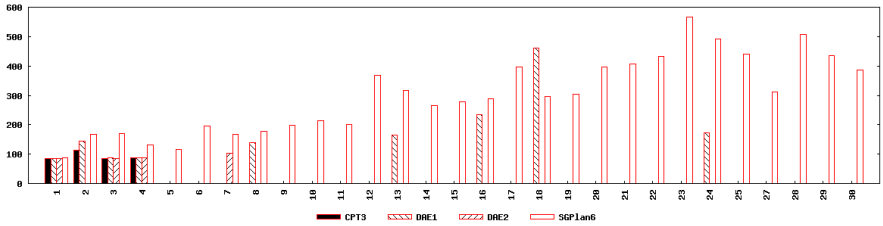

Fig. 3. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `openstack` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance).

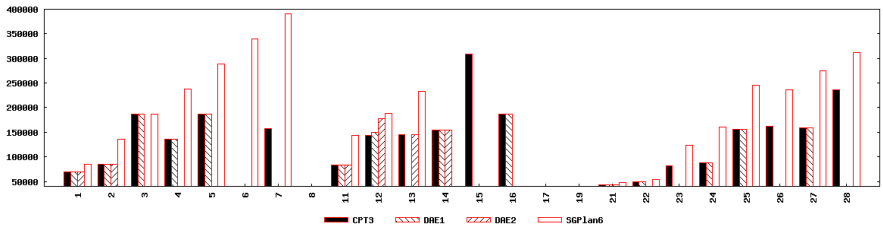

Fig. 4. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `parcprinter` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance).

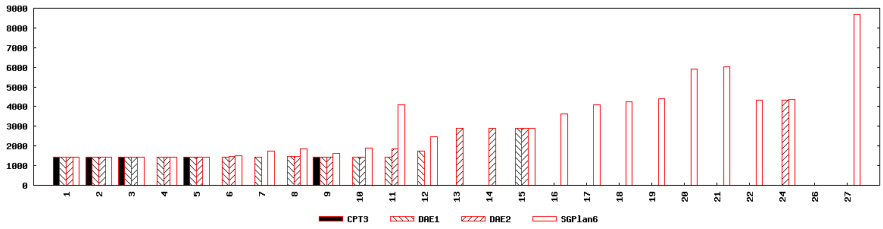

Fig. 5. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `crewplanning` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance).

Surprisingly, however, DAE failed to solve instances that CPT alone could solve, as `peg` 11 to 13 and 16 to 19, or `parcprinter` 7 to 15 (and even 3 to 5 for DAE2). First investigations of this strange behaviour showed that it was not due to the time limit, nor to a too small value for the maximal number of backtracks. It finally turned out that, for some of the random individuals of the first generation, CPT entered some infinite loop. This is again a case where Evolutionary Algorithms can be seen as "fitness function debuggers" [9].

5.3 A New Version of DAE

A patch was applied to CPT, and used within the DAE1 version, now termed DAE-p. Note that this patch modifies CPT internal consistency check, and could

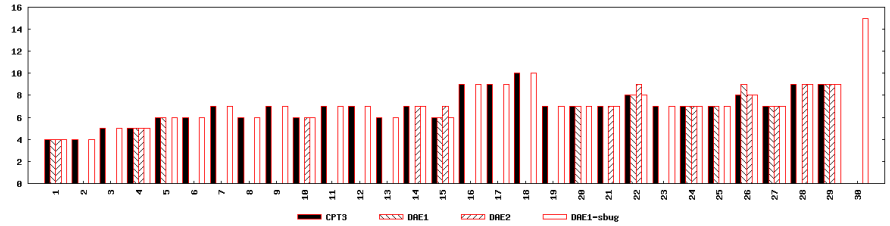

Fig. 6. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `peg solitaire` domain (one column per instance). No time limit.

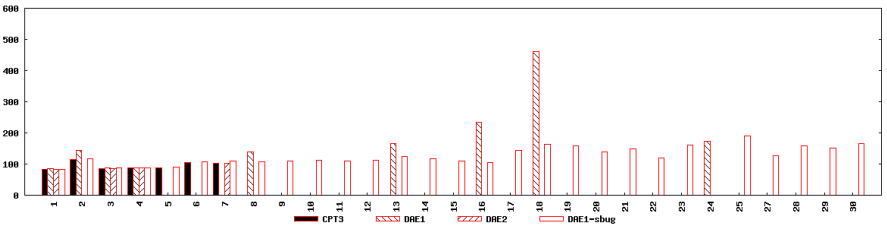

Fig. 7. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `openstack` domain (one column per instance). No time limit.

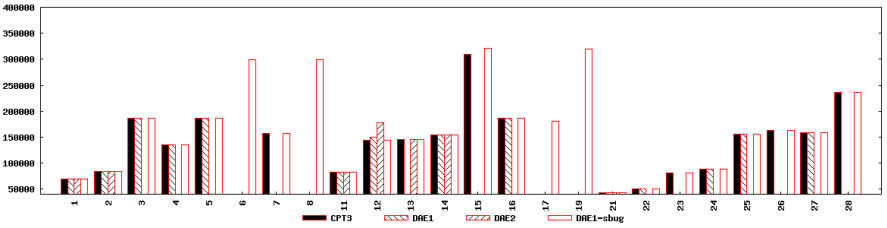

Fig. 8. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `parcprinter` domain (one column per instance). No time limit.

result in inconsistent plans. Hence all the solutions found by DAE-p were validated with the IPC solution checker (http://planning.cis.strath.ac.uk/VAL/), thus ensuring their consistency.

DAE-p was able to solve almost all instances of the 4 domains of the IPC-6 competition that CPT could handle, though demanding more than 30 min for the large instances: See Figures 6, 7, 8, and 9 for the comparative results of CPT, DAE1, DAE2 and DAE-p, and Figure 10 for examples of running times of the new DAE-p algorithm, where the horizontal bars show the 30min limit of IPC-6 competition: even DAE-p would have failed to solve the most difficult instances in the competition conditions.

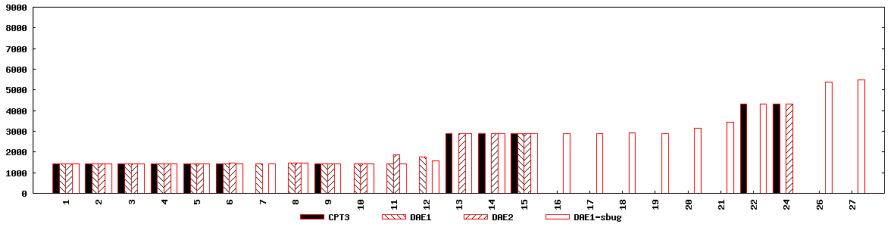

Fig. 9. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `crewplanning` domain (one column per instance). No time limit.

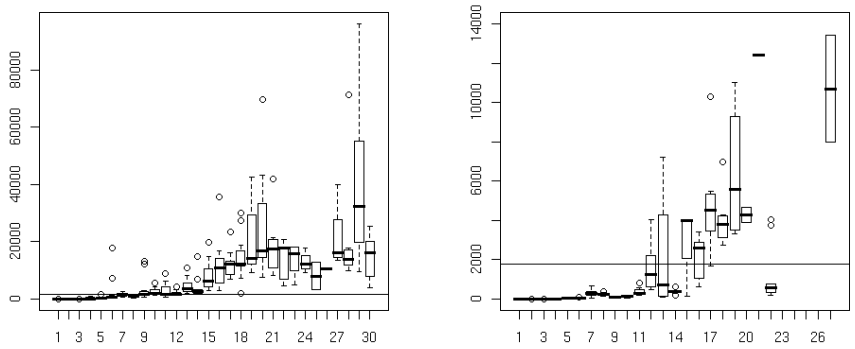

Fig. 10. Run times (s) of DAE-p on `openstack` (left) and `crewplanning` (right) domains. For each instance (column), the central box is the 25% – 75% quartile with median bar, the end of the upper (lower) dashed lines indicate the highest (lowest) values that are not considered as outliers, while the circles outside these lines are outliers.

5.4 Time-based Atom Choice

Another conclusion that can be drawn from the results of IPC-6 (Section 5.2) is that none of the planners DAE1 or DAE2 outperforms the other one. Indeed, though DAE1 outperforms DAE2 on most instances (e.g. for `parcprinter` domain Figure 8, or for instances `peg` 5, 20 and 25 Figure 6), the reverse is true on a large minority of other instances (e.g. `crewplanning` 24 Figure 9, `peg` 10, 14, 21, and 28 Figure 6). No satisfactory explanation could be found for this unpredictable behaviour. However, it demonstrates the need for a very careful choice of the atoms that are used to build the partial states.

This lead to propose a new method to actually build the partial states, based on a lower bound on the earliest time an atom can become true. Such lower bounds can easily be computed using a relaxation of the initial problem. The time before the earliest time all atoms from the goal can become true is then discretized into the number of partial states that are needed, and a partial state is build at each value of this discretized time by randomly choosing atoms between 1 and the number of subset of `mutex` atoms that are possible true at this time.

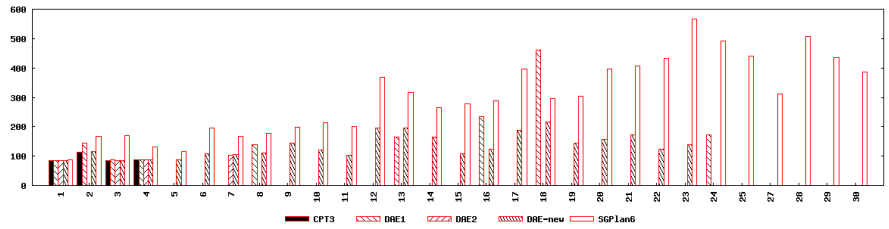

Fig. 11. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines), **DAE**$_{new}$ (grey), and SGPlan6 (the winner of the competition, white) on `openstack` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance). To be compared with Figure 3.

Variations operators (cf. Section 3.3) are modified changed in order take into account those earliest time for all atoms.

Preliminary results using this new algorithm, termed at the moment DAE$_{new}$, can be seen on Figure 11, and should be compared to those of Figure 3: DAE$_{new}$ solves all instances up to instance 23, and clearly outperforms SGPlan6 – though deeper and more intensive statistical tests are still on-going.

6 Conclusion and Further Work

Divide-and-Evolve is an original "memeticization" of Evolutionary and Operational Research algorithms in the area or Temporal Planning. A lot of progress has been made since its original inception [11], and it has now reached a level of performance high enough to be able to compete with the state-of-the-art planners of all kind.

First of all, DAE concept when using CPT as the embedded planner has been validated on many benchmarks of both IPC-3 and IPC-6 competitions, as DAE did solve many problems that CPT alone did not. Whereas this demonstrate the ability of DAE to overcome the curse of time complexity, DAE still fails when space-complexity is the issue, and CPT cannot even initialise the problem data.

DAE is the first evolutionary planner to have entered the IPC-6 competition, where CPT allowed DAE to tackle 4 out of 6 domains. A very positive result is that DAE gave a better makespan than SGPlan6, winner of the competition, whenever it could find a feasible solution (i.e. reach the goal). Unfortunately, there remained several instances that DAE could not solve – sometimes even without the harsh time limit of the competition, and even on instances that CPT alone could solve. This gave us an opportunity to detect a weird behaviour of CPT, that is unlikely to take place with 'standard' instances, but did happen with the random initial individuals of some evolutionary runs. The patched version of DAE was then able to solve most IPC-6 instances - though sometimes requiring much more CPU time than allowed during the official runs.

But there is still room for large improvements for DAE. First, the choice of the atoms that are used to represent individuals is still an open issue, and pre-

liminary experiments using more information from the planning domain are very promising. But another critical parameter is the maximum number of backtracks that we allow to CPT runs. The empirical formulae need to be refined, and this puts some light on the more general issue of parameter tuning: what is needed now are some descriptors of temporal planning instances, that would allow us to learn the best parameters based on the instance description. Such promising directions will be the subject of further research.

References

1. J. Bibai and M. Schoenauer and P. Savéant and V. Vidal. DAE: Planning as Artificial Evolution (Deterministic part). In IPC-2008 Competition Booklet (2008).
2. J. Bibai and M. Schoenauer and P. Savéant and V. Vidal. Evolutionary Planification by decomposition. INRIA Research Report No. RT-0355 (2008). http://hal.inria.fr/inria-00322880/en/
3. Y. Chen and C. Hsu and B. Wah. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *Artificial Intelligence*, 26, pages 323-369 (2006)
4. R. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3-4):189-208, (1971)
5. M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, Vol. 20, (2003)
6. A. Gerevini and A. Saetti and I. Serina. Planning through Stochastic Local Search and Temporal Action Graphs in LPG. *Journal of Artificial Intelligence Research*, 20, pages 239-290 (2003)
7. A. Gerevini and A. Saetti and I. Serina. On Managing Temporal Information for Handling Durative Actions in LPG. AI*IA 2003: Advances in Artificial Intelligence, Springer Verlag (2003)
8. D. Long and M. Fox. Exploiting a Graphplan Framework in Temporal Planning. Proceedings of ICAPS'03, pages 51-62 (2003)
9. F. Mansanne, F. Carrre, A . Ehinger, and M. Schoenauer. Evolutionary Algorithms as Fitness Function Debuggers. In Z. W. Ras and A. Skowron, Eds, *ISMIS'99*, LNCS 1609, Springer Verlag, 1999.
10. D. McDermott. PDDL – The Planning Domain Definition Language. At http://ftp.cs.yale.edu/pub/mcdermott (1998)
11. Marc Schoenauer and Pierre Savéant and Vincent Vidal. Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In J. Gottlieb and G. Raidl, Eds, *Proc. EvoCOP'06*, Springer Verlag (2006)
12. Marc Schoenauer and Pierre Savéant and Vincent Vidal. Divide-and-Evolve: a Sequential Hybridisation Strategy using Evolutionary Algorithms. In Z. Michalewicz and P. Siarry, Eds, *Advances in Metaheuristics for Hard Optimisation*, Springer Verlag, pages 179-198 (2007)
13. D. Smith and D. S. Weld. Temporal Planning with Mutual Exclusion Reasoning. Proc. IJCAI-99, pages 326-337 (1999)
14. V. Vidal and H. Geffner. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. Proc. of AAAI-2004, pp. 570-577 (2004)
15. V. Vidal and H. Geffner. Branching and Pruning: Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence* 170(3): 298-335 (2006).