On the Benefit of Sub-Optimality within the Divide-and-Evolve Scheme

Jacques Bibai, Pierre Savéant, Marc Schoenauer, Vidal Vincent

On the Benefit of Sub-Optimality within the Divide-and-Evolve Scheme

Jacques BIBAI[1 2], Pierre SAVÉANT[2],
Marc SCHOENAUER[1], and Vincent VIDAL[3]

[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
[3] ONERA – DCSD, Toulouse, France
Vincent.Vidal@onera.fr

Abstract. *Divide-and-Evolve* (DaE) is an original "memeticization" of Evolutionary Computation and Artificial Intelligence Planning. DaE optimizes either the number of actions, or the total cost of actions, or the total makespan, by generating ordered sequences of intermediate goals via artificial evolution. The evolutionary part of DaE is based on the *Evolving Objects* (EO) library, and can theorically use any embedded planner. However, since the introduction of this approach only one embedded planner has been used: the temporal optimal planner CPT. In this paper, we built a new version of DaE based on time-based Atom Choice and we embarked another planner (the sub-optimal planner YAHSP) in order to test the technical robustness of the approach and to compare the impact of using an optimal planner versus using a sub-optimal planner for all kinds of planning problems.

1 Introduction

An Artificial Intelligence (AI) planning problem is specified by the description of an initial state, a goal state, 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, whose execution from the initial state transforms it into a state that includes the goal state. The quality criterion of a plan depends on the type of available actions: number of actions in the simplest case; total cost for actions with cost; total makespan for durative actions which, in addition, may temporally overlap.

Domain independent planning is a fundamental and dynamic field of AI that has been tackled with a large amount of methods and algorithms, among which heuristic search (LAMA [15], FF [11], Fast Downward [10], YAHSP [19]), local search (LPG [7, 8]), and constraint programming (CPT [20, 21]). Among all these directions of planning research, and following some considerable successes of evolutionary algorithms in other areas of AI, Genetic Planning was introduced with the purpose to translate those success to planning problems. Introduced in

[12], several approaches to Genetic Planning have been proposed [18, 14, 22, 23, 4]. However, due to the limited performance of the resulting planning systems, the relevance of the possibility of application of EAs to planning was not deemed significant in comparison to the traditional methods.

However, an original approach, termed *Divide-and-Evolve* (DaE), was recently proposed [16, 17] to hybridize Evolutionary Algorithms (EAs) with Artificial Intelligence Planning. The baseline of DaE is to generate a sequence of partial states, thus dividing the initial problem into a sequence of subproblems, calling an external traditional planner to solve each subproblem in turn, and building a global solution from all subproblem solutions. DaE has participated to the last International Planning Competition (IPC) [2]: whereas hindered by the tight time limit, the quality of the solution plans it obtained on all the instances that it could solve in the temporal track was generally better than that of its competitors. Moreover, [1] demonstrates that those results can be improved for instance by a careful choice of the atoms that are used to build the partial states.

Until today, although DaE can theorically use any embedded planner, all experiments based on DaE have been performed using the optimal planner CPT as the embedded planner. The goal of this paper is to compare the performances of the DaE approach when using either CPT or a sub-optimal planner, such as YAHSP. The robustness of the hybrid algorithms and the quality of the solution plans they can find will be experimentally compared on all kinds of AI planning problems. Furthermore, the new version of DaE that will be used here builds on the time-based Atom Choice introduced in [1].

The paper is organized as follows: Section 2 briefly introduces planning problems; Section 3 recalls the *Divide-and-Evolve* approach, representation, fitness function and variation operators; Section 4 presents the results. The last section discusses results of DaE using either CPT or YAHSP as embedded planner, and sketches some directions for future work.

2 Planning Problems

Domain-independent planners rely on the Planning Domain Definition Language (PDDL) [13], inherited from the STRIPS model [5], to standardise and represent a planning problem. The language has been extended for representing temporality and action concurrency in PDDL2.1 [6].

The description of a planning problem splits into 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 state changes, 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 might be a partial state, i.e., it can be true in many

different (complete) 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, and possibly a duration or a cost. Preconditions are logical constraints which apply domain predicates to the arguments and trigger the effects when they are satisfied. Effects enable state transitions by adding or removing atoms.

A solution to a planning problem is a consistent schedule of grounded actions whose execution in the initial state leads to a state that contains one goal state, i.e., where all atoms of the problem goal are true.

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

In order to solve a planning problem $\mathcal{P}_D(I, G)$, the basic idea of DAE is to find a sequence of states $S_1, \ldots, S_n$, and to use some embedded 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 generation and optimization of the sequence of states (S_i) is driven by an evolutionary algorithm, and we will now describe its main components: the problem-specific representation of individuals, fitness, and variation operators.

3.1 Representation

As described in Section 2, a state is a list of atoms built over the set of predicates and the set of object instances. However, searching the space of complete states would result in a rapid explosion of the size of the search space. Moreover, goals of planning problem need only to be defined as partial states. It thus seems more practical to search only sequences of partial states, and to limit the choice of possible atoms used within such partial states. However, this raises the issue of the **choice of the atoms** to be used to represent individuals, among all possible atoms.

The result of the previous experiments on different domains of temporal planning problems from the IPC benchmark series [1] 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 build the partial states, based on the earliest time from which an atom can become true. Such time can be estimated by any admissible heuristic function (e.g h^1, h^2... [9]). The start times are then used in order to restrict the candidate atoms for each partial state. A partial state is then built at each time by randomly choosing among several atoms that are possibly true at this time. The sequence of states is then built by preserving the estimated chronology between atoms (**time consistency**). Heuristic function h^1 has been used for all experiments presented here.

Nevertheless, even when restricted to specific choices of atoms, the random sampling can lead to inconsistent partial states, because some sets of atoms can

be *mutually exclusive*[4] (`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. In practice, it is difficult to decide if several atoms are `mutex`. Nevertheless, binary `mutexes` can be approximated (i.e. not all pairs of mutually exclusive atoms can be discovered) with a variation of the h^2 heuristic function [9] in order to build quasi pairwise-mutex-free states (i.e., states where no pair of atoms are mutex).

An individual in the new version of DAE is hence represented as a variable length ordered time consistency list of partial states, and each state is a variable length list of atoms that are not pairwise `mutex`.

3.2 Fitness, and Embedded Planners

The fitness of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling an external 'embedded' 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, and up to now, only CPT has been used within the DAE approach. CPT [20, 21] is an exact planning system which combines a branching scheme based on Partial Order Causal Link (POCL) Planning with powerful and sound pruning rules implemented as constraints. But is optimality mandatory in order for DAE to obtain good quality results? In order to address this issue, a sub-optimal planner, YAHSP will be used here too. YAHSP [19] is a lookahead strategy planning system for sub-optimal STRIPS planning which uses the actions in the relaxed plan to compute reachable states in order to speed up the search process.

For any given k, if the chosen embedded planner succeeds in solving $\mathcal{P}_D(S_k, S_{k+1})$, the final complete state is computed by executing the solution plan from S_k, and becomes the initial state of the next problem. If all problems, $\mathcal{P}_D(S_k, S_{k+1})$ are solved by the chosen embedded planner, the individual is called *feasible*, and the concatenation of all solution 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 rescheduling some of its actions, in a step called *compression* (see [17] for detailed discussion). The quality of the compressed plan defines the fitness of a feasible individual.

However, as soon as the chosen embedded planner 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 the chosen embedded planner, as its initial state is in fact partially unknown, and no quality can be given to that individual. All such plans receive a penalty proportional to the number of subproblems solved such that the fitness of any infeasible individual is higher than that of any feasible individual.

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. It was necessary to limit the chosen local planner by adding some constraints in order to discard those subproblems.

[4] Several atoms are mutually exclusive when there exists no plan that, when applied to the initial state, yields a state containing them all.

And because, ultimately, it is hoped that all subproblems will be easy to solve, such limitation should not harm the search for solutions.

We have constrained CPT (resp. YAHSP) with a **maximal number of backtracks** (resp. a **maximal number of nodes**) that it is allowed to use to solve any of the subproblems. We have determined those bounds by a two-steps process: first, while evaluating the initial population, we allow a very large number of backtracks (resp. nodes) (e.g. 100000); the bounds are then chosen as the median of the actual number of backtracks (resp. nodes) that have been used to find the solutions during these evaluations of the initial population.

3.3 Initialization and Variation Operators

The initialization phase and the variation operators of the DAE algorithm respectively build the initial sequences of states and randomly modify some sequences during its evolutionary run.

The **initialization** of an individual is the following: first, the number of states is uniformly drawn between 1 and the number of estimated start times (see Section 3.1); For every chosen time, the number of atoms per state is uniformly chosen between 1 and the number of atoms of the corresponding restriction. Atoms are then chosen one by one, uniformly in the allowed set of atoms, and added to the individual if not `mutex` with any other atom already there.

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

Because an individual is a variable length list of states, and a state is a variable length list of atoms, the **mutation** operator can act here at two levels: at the individual level by adding (**addState**) or removing (**delState**) a state; or at the state level by adding (**addAtom**) or removing (**delAtom**) some atoms in the given state.

Note that the initialization process and these variation operators maintain the chronology between atoms in a sequence of states and the local consistency of a state, i.e. avoiding pairwise mutexes

3.4 Evolution Engine and Parameter Settings

A general issue in Evolutionary Computation (EC) lies in the number of parameters the programmer has to tune (from population size to selection operators to rates of applications of variation operators), and the lack of theoretical guidance to help him. Experimental statistical procedures have been proposed (e.g [24],[25]), that build on standard Design of Experiments methods and use the specificities of the EC domain to reduce the amount of computations.

In order to tune DAE, [3] proposed a two steps learning approach which involves choosing the probability and weights of each of the variation operators

being used with racing [24], and then choosing which predicates will be used to describe the intermediate goals with statistical analysis. In this paper we use the first step of [3] approach in several domains of IPC benchmarks and chose to keep the best common parameters configuration for all experiments of this paper.

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 50 generations, with a maximum of 1000 generations altogether. The probabilities of individual-level application of crossover and mutation (p_{cross} and p_{mut}) are (0.2, 0.8) and the relative weights of the 4 mutation operators ($w_{addState}$, $w_{delState}$, $w_{addAtom}$, $w_{delAtom}$) are (3,1,1,1).

4 Experimental Results

Divide-and-Evolve has been implemented within the Evolving Objects framework[5], an open source, template-based, ANSI C++-STL-compliant evolutionary computation library. In order to illustrate the behavior of DAE with each embedded planner, and to compare those implementations of DAE in all kind of planning problem, the following IPC benchmark domains have been used: `airport`, `satellite` and `logistics` domains for simple planning problem, `openstacks`, `scanalyser` and `woodworking` domains of the sixth IPC sequential satisficing track for planning with actions with costs, and `crewplanning`, `elevator` and `satellite time windows compiled` domains for temporal planning problem (actions with duration). Each domain has several instances of increasing complexity, hence a total of 470 problems.

Performance Measures:
All algorithms are given at most 2 hours of CPU time for each run on each problem instance. Their **efficiency** is then measured by the number of instances solved on each domain.

The quality of the plans are evaluated using IPC rules. For a given instance i, let Q_i^* be the best plan quality found among the competitor planners. The quality ratio for each planner is defined by Q_i^*/Q_i. The **quality score** of a planner for domain $\mathcal{D}$ is the sum over all instances of $\mathcal{D}$ of the quality ratios of this planner. The planner with the highest quality score is designated as the best performer on the domain. Note that if a planner cannot find a plan for a given instance after 2 hours, its quality ratio is set to 0 for this instance.

However, because $\text{DAE}_{\text{YAHSP}}$ and DAE_{CPT} are stochastic algorithms, 11 runs are performed on each instance in order to assess their robustness. Their **efficiency** per domain is defined as the total number of instances that have been

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

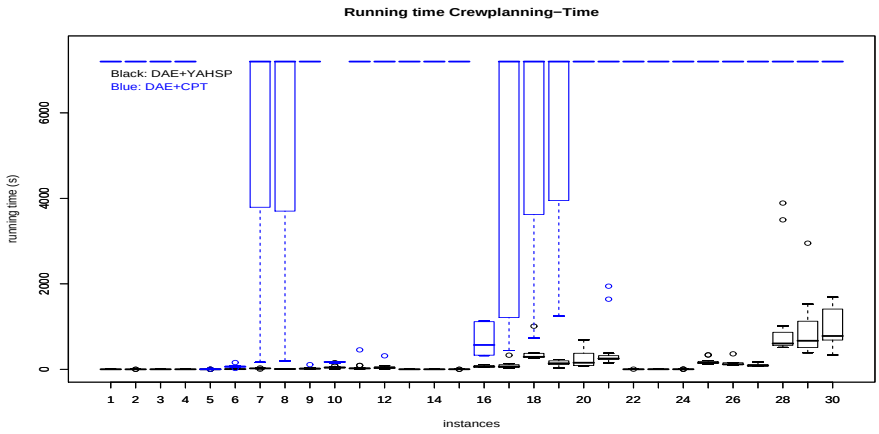

Fig. 1. Standard boxplots for the runtime distributions of DAE$_{\text{CPT}}$ (blue, on top) and DAE$_{\text{YAHSP}}$ (black, at bottom) on `crewplanning-Time` domain. The total runtime was bounded by 2 hours, hence the upper-limit of almost all boxplots for CPT.

solved at least once. The **average efficiency** for a given domain $\mathcal{D}$ is defined as $\frac{\sum_{i;n_i>0} n_i}{\sum_{i;n_i>0} 1}$, where n_i is the number of successful runs (i.e., that found a plan) for instance i of $\mathcal{D}$. It lies in $[0, 11]$. The **average quality** for domain $\mathcal{D}$ is defined as the sum over all solved instances i of $\mathcal{D}$ of $\frac{1}{n_i} \sum_{\{j \text{ solved } i\}} \frac{Q_i^*}{q_j}$ where q_j is the quality of the plan found by run j – the closer from the efficiency, the better.

Results:
First column (resp. second colum) of Table 1 shows for all algorithms the best efficiency $S_{planner}$ (resp. quality $Q_{planner}$) together with, in parentheses, the average efficiency (resp. average quality) for both DAE variants. Last column is the ratio $Q_{planner}/S_{planner}$. The mean values of those figures accross test domains are also provided, by domain category, and over all domains.

Figure 3 shows, for all algorithms, the plan quality of all instances accross 3 different domains (one of each category). Each column corresponds to an instance (number on the X axis). For the original planners YAHSP and CPT, symbols ('@' and '#' respectively) indicate the plan quality found. For both DAE variants, standard boxplots sketch the distribution of the qualities of the 11 plans. Figure 1 displays, for both DAE variants, the standard boxplots for the distribution of the 11 running times, and 2 shows one typical example of the fitness behavior along evolution on `elevator-Time` problem 2.

Discussion:
First, DAE$_{\text{YAHSP}}$ solves significantly more problems (79.36% of all problems) than YAHSP alone (71.49% of all problems), and much more than DAE$_{\text{CPT}}$ (21.70%) and CPT alone (10% only). Then, DAE$_{\text{YAHSP}}$ has the best quality score (see last line of Table 1) for all kinds of planning problem. Furthermore, DAE$_{\text{YAHSP}}$ consistently finds (see Table 1) either the optimal value, or a value

Fig. 2. Fitness behavior of DAE$_{\text{CPT}}$ (#) and DAE$_{\text{YAHSP}}$ (@) on the easy `elevator-Time` problem number 2.

more than 94% of the optimal value (as found by CPT) (Figure 3), and always finds a better plan quality than YAHSP alone (Figure 3 and Table 1). The running times of DAE$_{\text{YAHSP}}$, for instance on the `crewplanning` domain (Figure 1), are always smaller than those of DAE$_{\text{CPT}}$ (in fact, 2 hours was not enough for DAE$_{\text{CPT}}$). Thus, the variance of plan quality of DAE$_{\text{YAHSP}}$ (Figure 3) is generally smaller than that of DAE$_{\text{CPT}}$.

However, although the DAE$_{\text{YAHSP}}$ planner has the best sub-optimal ratio over all tested domains (last line of Table 1), DAE$_{\text{CPT}}$ has the best ratio on temporal domains (line 13 of Table 1). This is due to the quality of the compression step with the CPT constraint representation, where causal links and partial orders are inferred and exploited – which is not the case when YAHSP solved the subproblems. Nevertheless, there is no absolute best method here: Even in the case where one DAE variant obtains the best ratio value on a given type of problems, there is always at least one domain of this type where the other variant performs better on all instances it could solve (see table 1). See for instance, the `crewplanning` domain for temporal planning problems, and `airport` and `woodworking` domains for the other types of planning problem.

In all tested domains, DAE$_{\text{CPT}}$ (respectively DAE$_{\text{YAHSP}}$) could solve more problems than CPT (respectively YAHSP) alone. Furthermore, the average efficiency of both variants is very high (close to the maximum value 11), DAE$_{\text{YAHSP}}$ being slighly more robust than DAE$_{\text{CPT}}$: when an instance is solvable, almost all runs succeed. Regarding the quality robustness, the average quality of both variants are more often more than 90% of the quality score.

5 Conclusion

Divide-and-Evolve is an original "memeticization" of Evolutionary Computation and AI Planning. However, since the introduction of this approach, only one

embedded planner (the optimal CPT [20, 21]) had been used within DAE. The results presented in this paper, relying on the implementation of the time-based Atom Choice introduced in [1], demonstrate that

- it is indeed possible to embed in DAE another planner (here, the sub-optimal YAHSP [19]);
- when using the suboptimal YAHSP DAE can greatly improve the plan quality over that reached by YAHSP alone in all kind of planning problems;
- when embedding YAHSP, DAE is able to solve more problems within a 2 hours limit than when using the optimal planner CPT;
- the quality of the plans found by the $\text{DAE}_{\text{YAHSP}}$ version is very high, higher than that of DAE_{CPT} in simple and cost domains, and almost as good even in the case of temporal domains, where the compression step is much more efficient with the constraint-based CPT solver.

But there is still room for large improvements for DAE. First, it should be possible to define constraints in order to discard subproblems that are more difficult than the original global problem. This open issue could be addressed using for instance the empirical formulae of [1] that were designed to reduce the running time of DAE_{CPT} on temporal planning problems. Second, building on those results, it should be possible to combine several planners, taking advantage of the specificities of each of them by letting the Evolutionary Algorithm choose, for each subproblem, which planner to use. Such directions will be the subject of further research.

References

1. J. Bibai, P. Savéant, and M. Schoenauer. Divide-And-Evolve Facing State-of-the-Art Temporal Planners during the 6^{th} International Planning Competition. In C. Cotta and P. Cowling, editors, *EvoCOP'09*, pages 133–144. LNCS 5482, Springer-Verlag, 2009.
2. J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. DAE : Planning as Artificial Evolution (Deterministic part). At International Planning Competition (IPC) `http://ipc.icaps-conference.org/`, 2008.
3. J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. Learning Divide-and-Evolve Parameter Configurations with Racing. In A. Coles et al., editors, *ICAPS 2009, Workshop on Planning and Learning*, AAAI Press, 2009.
4. A. H. Brié and P. Morignot. Genetic Planning Using Variable Length Chromosomes. In *Proc. ICAPS*, 2005.
5. R. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 1:27–120, 1971.
6. M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20:61–124, 2003.
7. A. Gerevini, A. Saetti, and I. Serina. On Managing Temporal Information for Handling Durative Actions in LPG. In *AI*IA 2003: Advances in Artificial Intelligence*. Springer Verlag, 2003.
8. A. Gerevini, A. Saetti, and I. Serina. Planning through Stochastic Local Search and Temporal Action Graphs in LPG. *JAIR*, 20:239–290, 2003.

9. P. Haslum and H. Geffner. Admissible Heuristics for Optimal Planning. In *Proc. AIPS-2000*, pages 70–82, 2000.
10. M. Helmert. The Fast Downward Planning System. *JAIR*, 26(1):191–246, 2006.
11. J. Hoffmann and B. Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *JAIR*, 14:253–302, 2001.
12. J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
13. D. McDermott. PDDL – The Planning Domain Definition language. At `http://ftp.cs.yale.edu/pub/mcdermott`, 1998.
14. I. Muslea. SINERGY: A Linear Planner Based on Genetic Programming. In *ECP '97: Proceedings of the* 4^{th} *European Conference on Planning*, pages 312–324, London, UK, 1997. Springer-Verlag.
15. S. Richter, M. Helmert, and M. Westphal. Landmarks Revisited. In *Proc. AAAI'08*, pages 975–982. AAAI Press, 2008.
16. M. Schoenauer, P. Savéant, and V. Vidal. Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In J. Gottlieb and G. Raidl, editors, *Proc. EvoCOP'06*. Springer Verlag, 2006.
17. M. Schoenauer, P. Savéant, and V. Vidal. Divide-and-Evolve: a Sequential Hybridization Strategy using Evolutionary Algorithms. In Z. Michalewicz and P. Siarry, editors, *Advances in Metaheuristics for Hard Optimization*, pages 179–198. Springer, 2007.
18. L. Spector. Genetic Programming and AI Planning Systems. In *Proc. AAAI 94*, pages 1329–1334. AAAI/MIT Press, 1994.
19. V. Vidal. A Lookahead Strategy for Heuristic Search Planning. In 14^{th} *International Conference on Automated Planning & Scheduling - ICAPS*, pages 150–160, 2004.
20. V. Vidal and H. Geffner. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. In *Proc. AAAI*, pages 570–577, 2004.
21. V. Vidal and H. Geffner. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Artificial Intelligence*, 170(3):298–335, 2006.
22. C. H. Westerberg and J. Levine. "GenPlan": Combining Genetic Programming and Planning. In M. Garagnani, editor, 19^{th} *Workshop PLANSIG 2000)*, The Open University, 2000.
23. C. H. Westerberg and J. Levine. Investigations of Different Seeding Strategies in a Genetic Planner. In E. J. W. Boers et al., editors, *Applications of Evolutionary Computing*, pages 505–514, LNCS 2037, Springer-Verlag, 2001.
24. B. Yuan and M. Gallagher. Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms. In *Proc. PPSN VIII*, LNCS 3242, pages 172–181. Springer Verlag, 2004.
25. B. Yuan and M. Gallagher. Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In *Parameter Setting in Evolutionary Algorithms*, pages 121–142. Springer-Verlag, 2007.

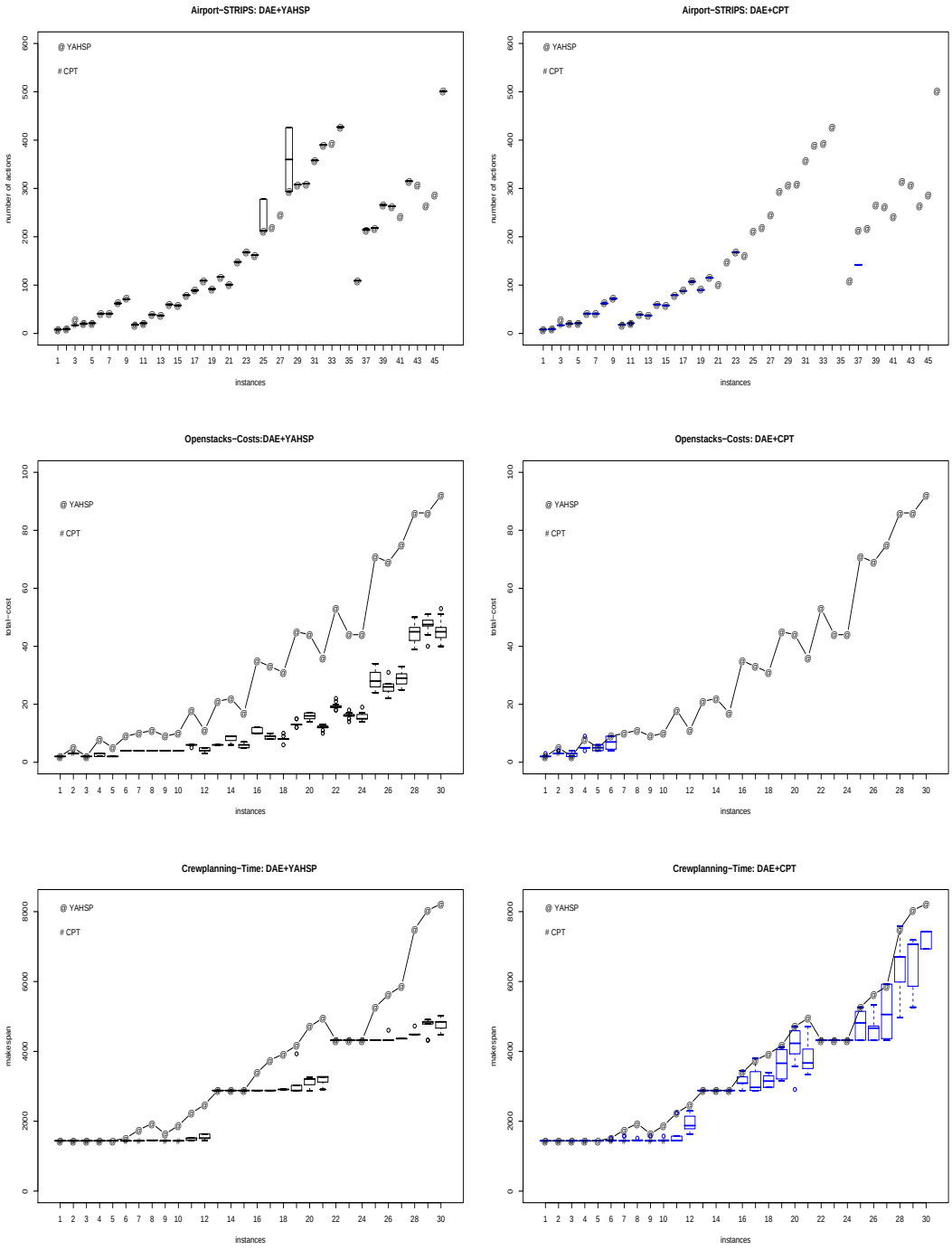

Fig. 3. Best plan quality found by CPT (#), YAHSP value (@) and the corresponding DAE variant (for each instance, one standard boxplot sketches the distribution of the 11 runs) on `aiport-STRIPS`, `openstacks-Costs` and `crewplanning-Time` domains. The boxplots follow standard usage: the central box 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.

Table 1. Quality and scaling of the optimal planner CPT, sub-optimal planners YAHSP, DAE$_{CPT}$ and DAE$_{YAHSP}$ accross the test domains. In column `Domain`(x), x denotes the total number of problem instances. Column 2-5 display the efficiency, i.e. number of instances solved (and also, for the DAE variants, the average number of successful runs – the closest to 11 the better). Column 7-10 show the quality score (and in parentheses, for DAE variants, the average efficiency, the closest to the quality score the better). See text for the exact definitions. The values in bold are the best values obtained on each type of planning problem (STRIPS, Cost, and Temporal). Column 11-15 displays the ratios $\frac{Quality\ Score}{Efficiency}$ on each domain (with means of those ratios accross the domain types). The underlined values in those columns are the best values obtained on each domain by the sub-optimal planners YAHSP, DAE$_{CPT}$ and DAE$_{YAHSP}$.

Domain	Efficiency				Quality				Quality/ Total of solved problems			
	YAHSP	CPT	DAE$_{YAHSP}$	DAE$_{CPT}$	YAHSP	CPT	DAE$_{YAHSP}$	DAE$_{CPT}$	YAHSP	CPT	DAE$_{YAHSP}$	DAE$_{CPT}$
Airport-STRIPS(50)	46	7	43 / 8.2	22 / 7.2	44.34	7	42.62 (42.1)	22 (22)	96.39%	100%	99.13%	100%
Satellite-STRIPS(36)	24	2	31 / 10.4	4 / 11	14.08	2	31 (30.8)	3.80 (3)	58.69%	100%	100%	95.24%
Logistics-STRIPS(198)	125	9	142 / 9.3	11 / 9.9	92.07	9	141.94 (133.5)	10.93 (10.9)	73.66%	100%	99.96%	99.45%
Total problems (284)	*195*	*18*	***216***	*37*	*150.50*	*18*	***215.56***	*36.74*	*76.25%*	***100%***	***99.70%***	*98.23%*
Openstacks-Cost(30)	30	3	30 / 10.9	6 / 6.8	11.60	3	30 (27.2)	5 (4.1)	38.70%	100%	100%	83.33%
Scanalyser-Cost(30)	27	3	28 / 10.5	6 / 7.7	16.48	3	27.81 (26)	5.92 (5.8)	61.04%	100%	99.35%	98.81%
Woodworking-Cost(30)	23	1	23 / 10.4	5 / 9	20.10	1	22.98 (22.3)	5 (3)	87.42%	100%	99.95%	100%
Total problems (90)	*80*	*7*	***81***	*17*	*48.19*	*7*	*80.80*	*15.92*	*62.39%*	***100%***	***99.77%***	*94.05%*
Crewplanning-Time(30)	30	14	30 / 10.9	30 / 9.5	24.65	14	29.97 (29.5)	29.01 (27.4)	82.19%	100%	99.93%	96.73%
Elevator-Time(30)	21	1	30 / 10.8	4 / 7	11.55	1	28.99 (23.6)	3.86 (3.6)	55.03%	100%	96.64%	96.74%
Satellite-Time(36)	10	7	16 / 9.2	14 / 8.1	7.47	7	15.15 (14.5)	14 (13.7)	74.78%	100%	94.72%	100%
Total problems (96)	*61*	*22*	***76***	*48*	*43.69*	*22*	***74.12***	*46.88*	*70.67%*	***100%***	*97.10%*	***97.82%***
Total (470)	336	47	**373**	102	242.39	47	**370.49**	99.56	69.77%	**100%**	**98.85%**	96.70%