On the Generality of Parameter Tuning in Evolutionary Planning

Jacques BIBAI [1,2]
Thales Research & Technology[2]
Palaiseau, France
firstname.lastname@thalesgroup.com

Pierre SAVÉANT
Thales Research & Technology[2]
Palaiseau, France

Marc SCHOENAUER
Projet TAO, INRIA Saclay,
& LRI-CNRS, Univ. Paris Sud[1]
& Joint lab Microsoft-INRIA
Orsay, France
firstname.lastname@inria.fr

Vincent VIDAL
ONERA – DCSD
Toulouse, France
Vincent.Vidal@onera.fr

ABSTRACT

Divide-and-Evolve (DAE) is an original "memeticization" of Evolutionary Computation and Artificial Intelligence Planning. However, like any Evolutionary Algorithm, DAE has several parameters that need to be tuned, and the already excellent experimental results demonstrated by DAE on benchmarks from the International Planning Competition, at the level of those of standard AI planners, have been obtained with parameters that had been tuned once and for-all using the Racing method. This paper demonstrates that more specific parameter tuning (e.g. at the domain level or even at the instance level) can further improve DAE results, and discusses the trade-off between the gain in quality of the resulting plans and the overhead in terms of computational cost.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

AI Planning, Memetic Algorithms, Parameter Tuning, Racing

1. INTRODUCTION

Parameter tuning is known as one of the main Achilles' heel of Evolutionary Algorithms (together with their computational cost). Whereas the efficiency of EAs has been demonstrated on several application domains (see for instance [33]), their successes were in general obtained through a tedious and time consuming parameter tuning phase. Furthermore, the newcomer to the field cannot benefit from theoretical guidance or recognized guidelines, and is then tempted to use off-the-shelf parameters, i.e. either default parameters of the framework he is using, or parameter values given in the literature for problems that resemble his.

Parameter setting, however, has today become an important field of research [19]. Following [9], one distinguishes between parameter tuning, that is done off-line, before running the algorithm, and parameter control, that is performed during the run, either based on the behavior of the algorithm (referred to as parameter adaptation), or relying on random modifications of the parameters together with the idea that fitness-based selection will also select adapted parameters (referred to as parameter self-adaptation). This paper is concerned with parameter tuning, i.e. off-line setting of the parameters of an Evolutionary Algorithm, in the domain of AI planning.

Several methods have been proposed for parameter tuning since Grefenstette's pioneering work [14]. However, they all face the same generalization issue: can a parameter set that has been optimized for a given problem be successfully used to some other problem? The answer of course depends on the similarity between both problems – and the issue then is to estimate the similarities between different optimization problems. However, even restricted to some precise optimization domain (like AI Planning), there are very few examples today of meaningful similarity measures between optimization problems, or, alternatively, of sufficiently precise and accurate features that would allow the user to describe the problem at hand with sufficient accuracy so that the optimal parameter set could be learned from this description, and carried on to other problems with similar description. The one exception we are aware of is the work of Hoos and co-authors in the SAT domain [17]: based on half a century of SAT work, and hundreds of papers, many relevant features have been gathered. Extensive parameter tuning on several thousands of instances have allowed the authors to learn a meaningful mapping between several parameterization of a given algorithm and the resulting performance of the algorithm (in terms of time to solution for satisfiable

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

SAT problems). Such mapping allows the user to come up with a good, if not optimal, set of parameters by applying this mapping to the features that can be easily computed for each unknown instance she/he has to solve.

This paper is concerned with such issues in the domain of the Artificial Intelligence (AI) Planning problems. An AI planning task written in PDDL (more details in Section 2.1) is specified by a *domain*, that defines the predicates and the actions, and an *instance*, that instantiates the objects, and specifies the initial state and a list of goals to reach. A solution is a plan, or a sequence of actions, such that when applied to the initial state it leads the system into a state where all goal atoms are true. Hence, for a given domain, there can exist several instances sharing the same predicates and actions, but differing either by the number of objects, or the initial and goal states.

Unfortunately, there does not exist any set of features for the AI Planning problem that are sufficient to describe the characteristics of a planning task, like mentioned above for the SAT domain [17]. The goal of this paper is hence less ambitious, though it can be seen as a first step in the direction of automatic parameter setting for the Evolutionary AI planner considered here. We will investigate the generality of behavioral parameters, i.e. parameters of the algorithm that directly impact on its behavior during the run (mainly the parameters related to the different variation operators, from their application rates to their internal parameters). We will also consider some structural parameters, namely here the set of predicates that are used to build the candidate solutions in the Evolutionary planner at hand, and analyze whether this set can be reduced, based on the analysis of previous runs, in order to reduce the size of the search space, and, hopefully, the computational cost of the optimization, without degrading the quality of the resulting solutions.

More practically, the experimental investigation proposed here will use 3 different domains of the AI Planning problem that have been proposed in the past International Planning Competitions (of 3 different types, see Section 4.1). Each domain contains a series of instances. Racing [7] will be conducted at a global level (aggregating quality from one instance from each domain), at the domain level (aggregating quality of the biggest instance of the domain), and at the instance level (an optimal parameter set will be determined for each instance). The way the performance of the algorithm decreases when going from instance to domain to global levels will give indications about both the homogeneity of the different domains and instances inside a domain, and the robustness of the parameter setting for the Evolutionary Planner at hand. Furthermore, the very good results that have already been obtained by DaE on those instances compared with the state-of-the-art have used a single parameter set (i.e. the result of global racing) [4]. The results presented here will assess that there is still room for improvement for the Evolutionary Planner DaE.

The paper is organized as follows: domain independent AI Planning is briefly introduced in Section 2.1. Section 2 presents DaE, the specific Evolutionary Planner. Section 3 rapidly recalls the basics of the Racing procedure that has been used here, as well as the experimental protocol. Experimental results are presented and analyzed in Section 4, comparing the performances of the parameter sets obtained with different levels of racing, and comparing the variation among the sets. As usual, Section 5 concludes the paper by sketching directions for further research.

2. DIVIDE AND EVOLVE

2.1 AI Planning

An Artificial Intelligence (AI) planning task is specified by the description of an initial state, a goal state, and a set of possible actions. An action can be applied only if certain conditions in the current state are met, and modifies the current state when applied. A solution to a planning task 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 (aka STRIPS domain); total cost for actions with cost; total makespan for durative actions which, in addition, may temporally overlap.

Domain-independent planners rely on the Planning Domain Definition Language (PDDL) [22], inherited from the STRIPS model [10], to standardize and represent a planning task. The language has been extended for representing temporality and action concurrency in PDDL2.1 [11]. The history of PDDL is closely related to the different editions of the International Planning Competitions (IPCs `http://ipc.icaps-conference.org/`), and the problems submitted to the participants are still the main benchmarks in AI Planning (see Section 4.1).

The description of a planning task splits into two separate parts: the generic *domain* on the 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, gives the initial state and provides a description of the goal. A *state* is described by a set of atomic formulae, or atoms. An atom is defined by a predicate symbol from the domain definition, 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 task 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 task defined on domain D with initial state I and goal G will be denoted $\mathcal{P}_D(I, G)$ in the following.

2.2 Evolutionary AI Planning

Early approaches to AI Planning using Evolutionary Algorithms directly handled possible solutions, i.e. possible plans: an individual is an ordered sequence of actions [28, 23, 31, 32, 8]. However, as it is often the case in Evolutionary Combinatorial optimization, those direct encoding approaches have limited performance in comparison to the traditional AI planning approaches, and hybridization with

classical methods had been the way to success in many combinatorial domains, as witnessed by the fruitful emerging domain of memetic algorithms [15]. Along those lines, though relying on an original "memeticization" principle, a novel hybridization of Evolutionary Algorithms (EAs) with AI Planning, termed *Divide-and-Evolve* (DAE) has been proposed [26, 27]. For a complete formal description, see [4].

In order to solve a planning task $\mathcal{P}_D(I, G)$, 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 tasks $\mathcal{P}_D(S_k, S_{k+1})$, 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.

2.3 DAE Representation

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 tasks from the IPC benchmark series [6] demonstrates the need for a very careful choice of the atoms that are used to build the partial states. The method used to build the partial states is based on an estimation of the earliest time from which an atom can become true. Such estimation can be obtained by any admissible heuristic function (e.g h^1, h^2... [16]). The possible start times are then used in order to restrict the candidate atoms for each partial state. A partial state is built at a given 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. However, these restrictions may still contain a large number of atoms, and it might be possible to further restrict this list only allowing atoms that are built with a restricted set of predicates. Manual choice had been used in the early versions of DAE [3]. However, it can be expected that such structural parameters can be learned by post-mortem analyzes of different runs of DAE on several problems of the same domain. This will be investigated in Section 4.

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*[1] (`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 seemed more efficient to try to a priori forbid them. In practice, it is not possible to decide if several atoms are `mutex` unless solving the complete problem. Nevertheless, binary `mutexes` can be approximated with a variation of the h^2 heuristic function [16] in order to build quasi pairwise-mutex-free states (i.e., states where no pair of atoms are mutex).

An individual in DAE is hence represented as a variable-length ordered time-consistent list of partial states, and each state is a variable-length list of atoms that are not pairwise `mutex`. Furthermore, all operators that manipulate the representation (see below) maintain the chronology between atoms and the local consistency of a state, i.e. avoid pairwise mutexes.

2.4 Initialization and Variation Operators

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 2.3); 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 other atoms that are 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. It is applied with probability p_{cross} to the individuals after selection.

After a possible crossover, an individual has a probability p_{mut} of being mutated. 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 4.1). 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. The choice between those operators is governed by user-defined relative *weights* ($w_{addStation}$, $w_{delStation}$, $w_{addAtom}$, $w_{delAtom}$). Furthermore, mutation addAtom also modifies every other atom of the state it is applied to with probability p_c, and, when adding a state, the possible new atoms are those that are possibly true in a time interval of radius r around the estimated time of the partial state at hand. r takes integer values, being an index in the array of possible times given by heuristic h^1 (see Section 2.3.

2.5 Fitness and Embedded Planners

In DAE, 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 can be used, and the early versions of DAE used the optimal planner CPT [30]. However, recent results [5] have demonstrated that much better results can be obtained in a more robust way by using a suboptimal planner, namely YAHSP [29], a lookahead strategy planning system for 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

[1] Several atoms are mutually exclusive when there exists no plan that, when applied to the initial state, yields a state containing them 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, in the case of temporal planning, this plan can in general be optimized by rescheduling some of its actions, in a step called *compression* in order to get a better makespan (see [27] for detailed discussion). The quality of the compressed plan defines the fitness of a feasible individual.

On the other hand, 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. All such plans receive a penalty inversely proportional to the number of solved subproblems, and such that the fitness of any infeasible individual is higher than that of any feasible individual.

Finally, in order to avoid the embedded planner to be stuck with subproblems that are in fact more difficult than the original one, YAHSP was constrained not to use more than a given number of nodes when solving any of the subproblems: first, a very large bound (e.g. 100000) is set when evaluating the initial population. The bound for the remaining of the run is then chosen as the median of the actual number of nodes that have been used to find the solutions during these evaluations.

3. PARAMETER TUNING FOR DAE

Two types of parameters are distinguished here: *structural* parameters act at the representation level, and include here the choice of the predicates that are used to build the intermediate states (see Section 2.3); Behavioral parameters directly impact on the way the optimization proceeds, and include the population size, the selection procedure and its parameters, plus all parameters related to the variation operators. However, the Darwinian-related parameters of DAE had been fixed after some early experiments [26, 27] (see Section 4.1), and only the 8 parameters related to the variation operators that have been described in Section 2.4 have been tuned using Racing.

Whereas tuning behavioral parameters pertains to standard Parameter Tuning procedure (Racing has been used here), problem-specific methods are required for the structural parameters. This Section details both tuning procedures in turn.

3.1 Learning Predicates Across Runs

The set of predicates that are used to describe the partial states of intermediate goals (see Section 2.3) is an important component of the representation, and has a large impact on the size of the search space. Hence pruning it as much as possible can become a crucial issue in very large domains. In order to assess the usefulness of a possible *Learning across runs* technique, where, within a given domain, some runs could help identifying the useful predicates, the following statistics were gathered during the experiments. 50 runs were run on the first 11 instances of each domain, using the parameters learned at the domain level. All atoms of the 11 $\times$ 50 best plans were gathered, and two figures were computed for each predicate: First, the frequency of appearance, i.e. the proportion of those optimal plans that did contain this predicate (number in $[0, 1]$ for each predicate, 1 would mean that this predicate was present in every solution plan); Second, the proportion of atoms that did contain this predicate – all those proportions sum up to 1.

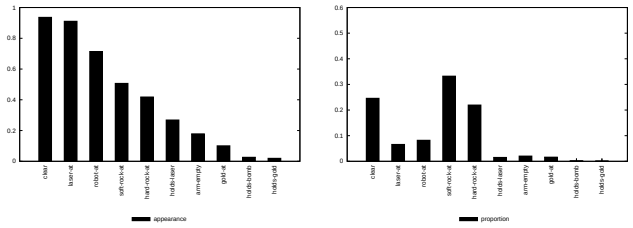

Figure 1: Predicate statistics: Frequency of occurrence (left) and proportion in all atoms (right) for `gold-miner` domain. The predicates are in the same order on the x-axis of both plots.

Based on those statistics, some clustering method was applied to this dataset. Relational analysis [2] with a threshold of similarity equal to 0.5 was chosen: initiated and developed at IBM in the 70s [20], relational analysis does not require to arbitrarily choose the number of clusters to be discovered. After this analysis, the predicates that belong to classes with more than 2 elements are retained, with the expectation that restraining the choice of predicates for the resolution of the other instances (not the first 11 ones, that were used to identify those predicates) would speed up the search (because the search space is smaller) while maintaining the same quality of the solution.

This approach was tried on several domains, and Figure 1 displays both statistics described above for the `gold-miner` domain, using the experimental settings described in next Section. Both statistics obviously capture different criteria of importance for the predicates. Unfortunately, the results obtained by DAE when using the restricted set of predicates chosen according to either of those statistics were not better, and sometimes even worse, than the results of DAE using the complete set of predicates. Furthermore, even when the results were similar in quality, they were not obtained significantly faster. Based on those experiments, it is not clear that a restricted set of predicates can be learned, and at least it cannot be learned as proposed here: the automatic choice of a restricted set of predicates is still an open issue, subject of further work.

3.2 Racing Behavioral Parameters

Since the early days of Evolutionary Algorithms [14], researchers have tried to optimize the control parameters of their favorite algorithms. Classical statistical methods like ANOVA (Analysis of Variance), based on some extensive DOE (Design Of Experiment) can be used out-of-the-box: a finite number of parameter sets is chosen (e.g. a factorial design that includes all possible combinations of a finite number of values for each parameter), and a given number of runs is run for each set. Standard ANOVA test indicates whether some statistically significant difference exists among all the sets, and pairwise tests then designate the best set.

Originally proposed for solving the model selection issue in Machine Learning [21], Racing techniques were introduced in Evolutionary context [7] in order to decrease the computational cost of such naive approach by rapidly focusing the search on the most performing parameter configurations. The basic idea of Racing techniques is to identify, with a given statistical confidence level, the poorly-performing parameter configurations after only a few runs, and to go on running only the promising configurations: after each

run, all configurations are tested against the best one, and the ones that are significantly worse are simply discarded. Such cycle execution-comparison-elimination is repeated until only one parameter configuration remains, or the total number of experiments has been run.

The efficiency of such technique highly depends on the statistical test used for the comparison. Because no assumption can be made about the distribution of the results (e.g. normality), the most popular races [7, 34, 35] are based on the non-parametric Friedman's two-way analysis of variances by ranks. Furthermore, given the statistical test, the user must set two parameters for Racing: the number of initial runs before starting the comparisons, and the confidence level of the test.

The performance used for the comparison is generally the best fitness reached for a given stopping criterion. Here, in order to slightly favor the algorithms that run faster, a secondary component was added to the performance measure: the fitness takes only integer values (however much the type of problem, see Section 4.1), and some penalty in [0,1] was hence added to the fitness. This penalty is proportional to the computational time of the algorithm (normalized by the maximum allowed time).

4. EXPERIMENTAL RESULTS

4.1 Experimental Setting

Previous experiments demonstrate that DAE can greatly increase the performance of an encapsulated suboptimal planner, both in terms of coverage and solution quality, making it competitive with state-of-the art planners [4] even when it have used a single parameter set for all problems.

In order to show that there is still room for improvement for the Evolutionary Planner DAE, experiments have been run on 3 domains, and represent 3 different types of problems: `gold-miner` is a STRIPS problem (the quality of a plan is the number of actions) from the IPC6 learning track, `openstacks` uses actions with costs (the quality of a plan is the total cost) of the IPC6 sequential satisficing track, and `zeno` is a temporal problem (the quality of a plan is the total makespan) from IPC3. All are available from IPC web site at `http://ipc.icaps-conference.org/`.

Furthermore, in addition to comparing the results of the different parameter sets obtained by the different parameter tuning procedures described in Section 3, DAE results have been compared to those of the best learner in their category: the optimal CPT [30] for STRIPS domain, LPG [12, 13] for temporal domain, and LAMA [25] (updated version kindly given by the authors) for actions with costs. As is the case for CPT, however, those learners were given a strict 30min time limit. In particular, this explains why the best learner is not always CPT, which is an optimal learner, and could find the optimal value for most instances if given enough time (though it cannot solve the largest `zeno` instances whatever the time it is given).

DAE has been implemented within the Evolving Objects framework (`http://eodev.sourceforge.net/`), an Open Source, STL-based, C++ Evolutionary Computation library. The fixed *evolution engine* is a (100+700)-ES: 100 individuals generate 700 offsprings without selection. The survival selection is performed among those 800 individuals using a *deterministic tournament* of size 5. For all runs, the **stopping criterion** is the following: After at least 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. Furthermore, all runs were allowed a maximum CPU time of 30mn (running on 3.4 GHz cores).

For the Racing procedure (see Section 3.2), the initial number of runs was set to 11 (the lowest number for significance of the statistical test), and the confidence level to 0.025 (strong constraint for the rejection of equality hypothesis between two parameter configurations). The racing process was stopped after at most 50 runs.

For the 8 behavioral parameters (Section 2.4), 2 values were tried, giving a total of 256 different parameter configurations (more configurations would have resulted in too long experiments). The following values were used: $p_{cross} = 0.2$ or 0.6, $p_{mut} = 0.6$ or 0.8, $w_* = 1$ or 3, the relative weights of the 4 mutation operators, $r = 1$ or 2, the tolerance radius for time consistency, and $p_c = 0.2$ or 0.8, probability to modify an atom in a state undergoing mutation.

The global racing was performed by aggregating the performances over one instance per domain, namely `goldminer30`, `openstacks21`, and `zeno14`. For the racing per domain, the instance with highest index (supposedly the most difficult one) was chosen, namely `goldminer30`, `openstacks30`, and `zeno20`.

4.2 Best Behavioral Parameters

Whereas $336 \times 50 = 12080$ runs would have been run by a full factorial DOE, the global racing needed only 4351 runs, and racing per domain 4013, 4021 and 3966 runs for respectively `gold-miner`, `zeno simple time`, and `openstacks` domains. the full per instance racing required on average 5259, 5817 and 5473 runs for respectively `gold-miner`, `zeno simple time`, and `openstacks` domains. However, those averages hide large variations: paradoxically, all easy instances required the maximum number of runs (12080), as several parameter configurations could solve the instance to optimum. This was the case for instances 1-20 for `gold-miner`, 1-12 for `zeno` and 1-18 for `openstacks`. Racing did save around 65% CPU time for the global or per-domain versions, and little less for the racing per instance - though we could have saved doing racing on the easy instances, as the comparative results below will confirm.

Regarding the best parameter configurations, Figure 2 displays them all, on 8 lines: the horizontal line represents the value obtained by the global racing, and only the values that differ from this one are marked for each instance (column) or the racing per domain (last column of each plot). There seems to be a large consensus on the highest value for p_c (0.8, second line from bottom), that matches the results of the global racing and all 3 domain racings. There is, too, a reasonable consensus for choosing $r = 2$ (bottom line) for `gold-miner` and `zeno` domains, matching the result of the global racing, too, but `openstacks` racings prefered the other value. On the other extreme, the values retained by the instance-racings for p_{cross}, p_{mut}, $w_{delState}$, $w_{delAtom}$, and $w_{addState}$ (lines 3-7 from bottom) seem to contradict those of the global racing, while matching those of the domain-racing only very partially. It seems that even when restricting the number of possible values of the behavioral parameters to 2, different instances even belonging to the same domain are globally quite heterogenous for DAE.

4.3 Comparing Racing Procedures

Figure 3 displays the box-plots of the 50 runs for each domain (top to bottom), each racing level (left to right), and each instance (the x-axis). Additionally, the results of YAHSP alone (the embedded planner), as well as the results of the best opponent in its category. The numerical values of the means and minimal values reached are also given, for the sake of completeness, in the large unlabeled table (for space reasons) at the end of the paper.

An immediate conclusion is the confirmation that DAE outperforms by far YAHSP alone (as already demonstrated in [5]). A second obvious conclusion is that the performances of DAE when doing one racing per instance are better than the other racing options. However, this is particularly true for the most difficult instances (the instances with the highest index, at least), and for domains `zeno` and `openstack`, whereas `gold-miner` seems easier to solve to optimality, or near-optimality, with either global or per-domain racing: only slight differences for few instances can be reported for the best achieved values. However, the variation among runs, as witnessed by the height of the box-plots of Figure 3, are clearly smaller with racing per instance... some difference that is not so large on both other domains. Surprisingly, on several instances (e.g. `zeno 14, 15, 18` and `openstacks 24, 25, 28`, the global racing found better results (lower minimum, similar median) than the per-domain racing. This seems to indicate that those domains are rather heterogeneous in difficulty for the large instances.

Finally, one good news, on the other hand, is that DAE is able to find equal or better results than its best opponent in the experimental conditions chosen here, as witnessed by the last column of the final table: for some large instances, both LAMA and LPG, at least when limited to the same harsh 30min constraint that was imposed here, are frequently outperformed by DAE – at the cost of an instance-based racing to tune DAE parameters.

5. CONCLUSION

DAE is an original "memeticization" of Evolutionary and planning algorithms in the area of AI Planning. A lot of progress has been made since its original inception [26, 27], and it has now reached a level of performance that is able to compete with the best state-of-the-art planners even when it uses one single parameter configuration for all problems [4]. In this paper, however, the results on three different types of IPC problems showed that the quality of the results obtained with DAE can be further improved with more specific parameter tuning, although these results were not uniform over all tested domains. The practicality of such approach remains however questionable, as racing per instance requires to solve around 5000 times the instance with different parameter configurations. Racing per domain is an alternative whenever the user has to repeatedly solve new instances from the same domain. However, referring to the values retained by the different procedues (as discussed in Section 4.2), the different instances even of the same domain require different parameter settings, and the costly instance-based racing seems to be mandatory to get the best restults on the large and difficult instances.

Nevertheless, there is still room for improvement in tuning DAE's parameters. The choice of the set of parameter configurations for the racing is still an open issue. Furthermore, it was limited here by the CPU cost, and only 256 different parameter configurations could be tried. There are other alternatives to racing, that have the advantage that they actually optimize the parameters, i.e., are not restricted to a pre-defined set of configurations.

For instance, the Sequential Parameter Optimization method (SPO) [1] alternatively tries to build a model of the performance of the algorithm as a function of the parameters using Gaussian Kernels. Some improvements to SPO have already been proposed [18], that reduce its computational overhead and increase its accuracy. However, it remains to be compared to Racing for the Evolutionary Planning with DAE. Also, recently, the REVAC method has been proposed [24], that uses as a meta-EDA (Estimation of Distribution Algorithm) to optimize the parameters of an EA, but also estimates the relevance of each parameter at hand. This could confirm some results obtained here, where some parameters seem irrelevant because most racing results proposed the same value.

But the ultimate step in the direction proposed in this paper would be to design a completely automated procedure for parameter tuning, that would not require extensive runs that are unaffordable in real world situations for instance. The way toward this grail probably lies in Adaptive or Self-Adaptive techniques, yet to be proposed outside the continuous domain where such techniques have now reached maturity [9].

6. REFERENCES

[1] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. Sequential parameter optimization. In B. McKay, editor, *Proc. CEC'05*, pages 773– 780. IEEE Press, 2005.

[2] H. Benhadda and F. Marcotorchino. L'analyse relationnelle pour la fouille de grandes bases de données. *In Revue des Nouvelles Technologies de l'Information*, RNTI-A-2, 2007.

[3] 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.

[4] J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *ICAPS 2010*, pages 18–25. AAAI press, 2010.

[5] J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. On the benefit of sub-optimality within the divide-and-evolve scheme. In P. Cowling and P. Merz, editors, *EvoCOP 2010*, number 6022 in Lecture Notes in Computer Science, pages 23–34. Springer-Verlag, 2010.

[6] J. Bibai, M. Schoenauer, and P. Savéant. Divide-And-Evolve Facing State-of-the-Art Temporal Planners during the 6^{th} International Planning Competition. In C. Cotta and P. Cowling, editors, *Ninth European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2009)*, number 5482 in Lecture Notes in Computer Science, pages 133–144. Springer-Verlag, 2009.

[7] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. In *GECCO '02*, pages 11–18. Morgan Kaufmann, 2002.

[8] A. H. Brié and P. Morignot. Genetic Planning Using Variable Length Chromosomes. In *Proc. ICAPS*, 2005.

[9] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In Lipcoll et al. [19], chapter 2, pages 19–46.

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

[11] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20:61–124, 2003.

[12] A. Gerevini, A. Saetti, and I. Serina. On Managing Temporal Information for Handling Durative Actions in LPG. In *AI*IA*

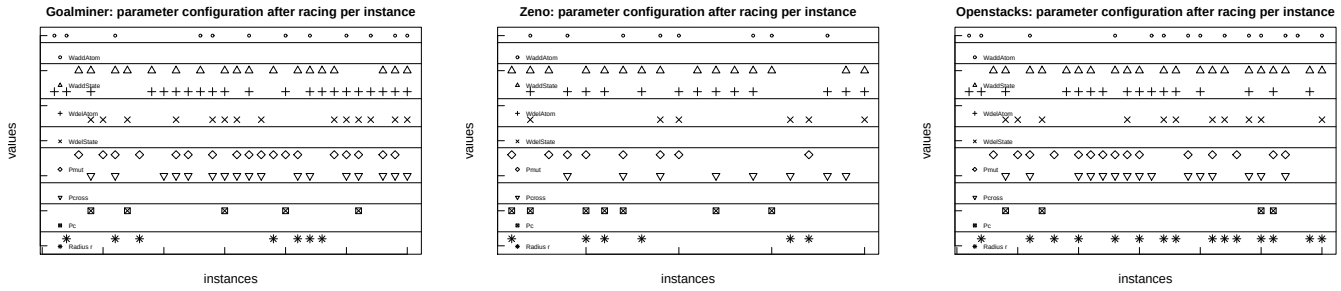

Figure 2: Best parameter configurations after racing per instance, for `gold-miner`, `zeno` and `openstacks` domains. From top to bottom, $w_{addAtom}$, $w_{addState}$, $w_{delAtom}$, $w_{delState}$, p_{mut}, p_{cross}, p_c, r, and each parameter could take 2 values (see text). For each of those parameters, the horizontal line represents the value retained by the global racing, the last column shows that retained by the racing per domain, and each other column is the value obtained from racing on a single instance.

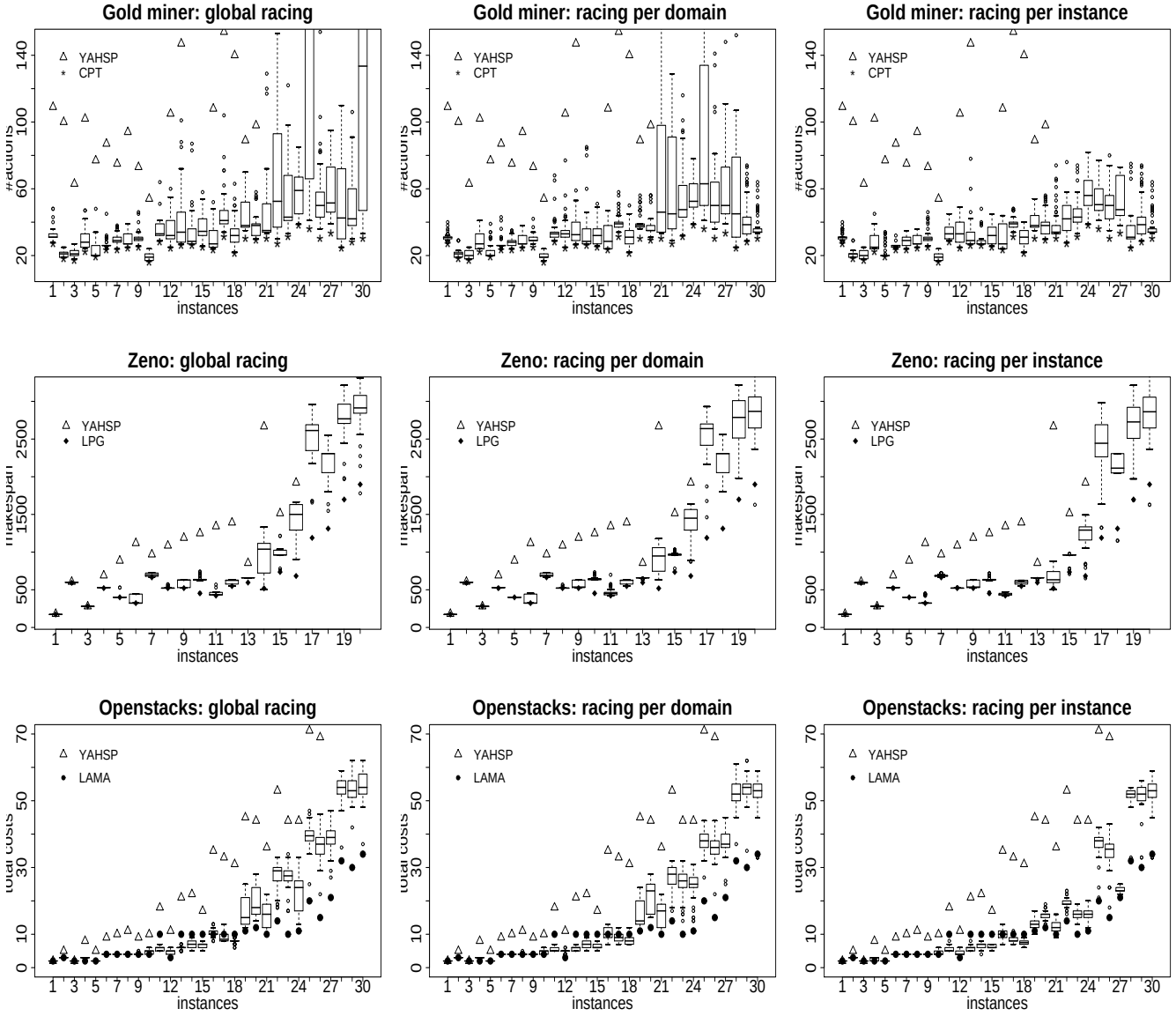

Figure 3: Results for DAE on `gold-miner`, `zeno` and `openstacks`, together with the value found by YAHSP alone (△), and the value from the best competitor, CPT (*), LPG (⧫), or LAMA (•). Note that YAHSP did solve all instances, though the poor quality of its solution prevents from displaying it for the sake of readibility.

2003: Advances in Artificial Intelligence. Springer Verlag, 2003.

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

[14] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-16, 1986.

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

[16] P. Haslum and H. Geffner. Admissible Heuristics for Optimal Planning. In *Proc. AIPS-2000*, pages 70–82, 2000.

[17] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *CP 2006*, number 4204 in lncs, pages 213–228. Springer Verlag, 2006.

[18] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In Franz Rothlauf et al., editor, *GECCO'09*, pages 271–278. ACM, 2009.

[19] F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, Berlin, 2007.

[20] F. Marcotorchino and P. Michaud. Agrégation des similarités en classification automatique. *In Revue de Statistique AppliquÃl'e*, Vol 30-No 2, 1981.

[21] O. Maron and A. W. Moore. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In *NIPS 6*, pages 59–66. Morgan Kaufmann, 1994.

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

[23] I. Muslea. SINERGY: A Linear Planner Based on Genetic Programming. In *Proc. ECP '97*, pages 312–324. Springer-Verlag, 1997.

[24] V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In M. Veloso, editor, *Proc. Intl. Joint Conference on Artificial Intelligence*, pages 975–980, 2007.

[25] S. Richter, M. Helmert, and M. Westphal. Landmarks Revisited. In *AAAI'08*, pages 975–982. AAAI Press, 2008.

[26] 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.

[27] 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.

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

[29] V. Vidal. A Lookahead Strategy for Heuristic Search Planning. In 14^{th} *International Conference on Automated Planning & Scheduling - ICAPS*, pages 150–160, 2004.

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

[31] C. H. Westerberg and J. Levine. "GenPlan": Combining Genetic Programming and Planning. In M. Garagnani, editor, *19th PLANSIG Workshop*, 2000.

[32] C. H. Westerberg and J. Levine. Investigations of Different Seeding Strategies in a Genetic Planner. In E.J.W. Boers et al., editor, *Applications of Evolutionary Computing*, pages 505–514. LNCS 2037, Springer-Verlag, 2001.

[33] T. Yu, L. Davis, C. Baydar, and R. Roy, editors. *Evolutionary Computation in Practice*. Studies in Computational Intelligence 88, Springer Verlag, 2008.

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

[35] 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.

#	Global racing		Domain racing		Instance racing		Best
Gold	Min	med.	Min	med.	Min	med.	CPT
1	**28**	31	**28**	31	**28**	31	28
2	**19**	21	**19**	21	**19**	21	19
3	**18**	21	**18**	20	**18**	20	18
4	**24**	28	**24**	27	**24**	25	23
5	**20**	20	**20**	20	**20**	20	20
6	**24**	26	**24**	26	**24**	26	24
7	**24**	29	**24**	28	**24**	28	24
8	**25**	27	**25**	27	**25**	27	25
9	**26**	30	27	29	**26**	29	26
10	**17**	19	**17**	19	**17**	19	17
11	**29**	33	**29**	33	**29**	33	29
12	**25**	32	**25**	32	**25**	33	25
13	**27**	34	**27**	32	**27**	29	27
14	**27**	28	**27**	28	**27**	27	27
15	**26**	34	**26**	32	**26**	32	26
16	**24**	27	**24**	27	**24**	27	24
17	**34**	41	**34**	39	**34**	39	32
18	**22**	32	**22**	31	**22**	30	22
19	**35**	38	36	38	**35**	38	31
20	**30**	38	**30**	35	**30**	38	30
21	**33**	35	**33**	46	**33**	34	31
22	30	51	29	43	**28**	41	28
23	33	43	**32**	46	33	43	32
24	**39**	55	**39**	52	**39**	52	39
25	**41**	173	**41**	62	**41**	50	37
26	36	50	36	50	**35**	50	31
27	39	50	39	49	**38**	46	34
28	**25**	40	**25**	41	**25**	36	25
29	30	41	**29**	38	**29**	38	29
30	**33**	130	**33**	34	**33**	34	31
zeno	Min	med.	Min	med.	Min	med.	LPG
1	**173**	173	**173**	173	**173**	173	173
2	**592**	599	**592**	599	**592**	592	592
3	**280**	280	**280**	280	**280**	280	280
4	529	529	**522**	529	**522**	529	522
5	**400**	400	**400**	400	**400**	400	400
6	**323**	323	**323**	323	**323**	323	323
7	672	692	**665**	692	**665**	679	665
8	**522**	522	**522**	522	**522**	522	522
9	**522**	629	**522**	536	**522**	536	522
10	**453**	636	**453**	636	**453**	636	453
11	433	433	**423**	453	**423**	433	423
12	**549**	626	**549**	616	**549**	603	549
13	659	659	633	659	**626**	659	596
14	**503**	1009	633	905	**503**	633	519
15	756	962	782	962	**709**	962	736
16	906	1438	872	1432	**653**	1292	683
17	1658	2454	1462	2623	**1324**	2444	1189
18	1547	2304	1801	2300	**1152**	2114	1312
19	1968	2740	1700	2767	**1691**	2710	1698
20	1780	2900	**1628**	2860	**1628**	2860	1898
Open	Min	med.	Min	med.	Min	med.	LAMA
1	**2**	2	**2**	2	**2**	2	2
2	**3**	3	**3**	3	**3**	3	3
3	**2**	2	**2**	2	**2**	2	2
4	**2**	3	**2**	3	**2**	3	2
5	**2**	2	**2**	2	**2**	2	2
6	**4**	4	**4**	4	**4**	4	4
7	**4**	4	**4**	4	**4**	4	4
8	**4**	4	**4**	4	**4**	4	4
9	**4**	4	**4**	4	**4**	4	4
10	**4**	5	**4**	5	**4**	4	4
11	**5**	6	**5**	5	**5**	5	10
12	**3**	5	**3**	5	**3**	4	3
13	**5**	6	**5**	6	**5**	6	10
14	5	7	5	7	**4**	7	10
15	**5**	7	**5**	6	**5**	6	10
16	8	10	**7**	10	**7**	10	10
17	8	8	**7**	9	**7**	8	10
18	**6**	8	7	8	**6**	7	18
19	12	14	11	14	**10**	13	11
20	14	17	15	22	**12**	16	12
21	10	16	10	16	**9**	12	10
22	18	29	17	28	**16**	19	14
23	17	27	14	26	**13**	14	10
24	13	24	14	25	**12**	16	11
25	25	39	27	38	**21**	38	20
26	22	36	22	36	**18**	35	15
27	27	39	25	37	**22**	25	21
28	36	53	37	52	**33**	52	32
29	42	53	35	54	**33**	54	30
30	37	54	33	53	**33**	53	34