Learning *Divide-and-Evolve* Parameter Configurations with Racing

Jacques Bibaï[1,2] **Pierre Savéant**[1] **Marc Schoenauer**[2] **Vincent Vidal**[3]
[1]Thales Research & Technology, Palaiseau, France, firstname.lastname@thalesgroup.com
[2]Projet TAO, INRIA Saclay & LRI, Université Paris Sud, Orsay, France, marc.schoenauer@inria.fr
[3]CRIL & Université d'Artois, Lens, France, vidal@cril.univ-artois.fr

Abstract

The sub-optimal *Divide-and-Evolve* (DAE) planner implements the stochastic approach for domain-independent planning decomposition introduced in (Schoenauer, Savéant, and Vidal 2006; 2007). 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 the embedded planner is the non-optimal STRIPS planner YAHSP (Vidal 2004). For a given domain, the learning phase uses a 'racing' procedure to choose the rates of the different variation operators used in DAE, and processes the results obtained during this process to specify the predicates that will be later used to describe the intermediate goals.

Introduction

Divide-and-Evolve (DAE) is a generic hybrid approach to solve *Planning Problems* (PPs), originally introduced in (Schoenauer, Savéant, and Vidal 2006; 2007). It uses an evolutionary algorithm to evolve an ordered sequence of subgoals; the resulting subproblems (going from one subgoal to the next) are then submitted on to an embedded planner; if all these PPs are solved, the concatenation of all corresponding subplans is a solution of the initial PP. The makespan (or number of actions or the total cost of actions) of this solution defines the fitness of the sequence of subgoals used by the evolutionary algorithm.

A general issue in Evolutionary Computation (EC), that somewhat hinders its wide use in spite of some highly successful applications, 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, that build on standard Design of Experiments methods and use the specificities of the EC domain to reduce the amount of computations. Among those, the *racing* approach (Yuan and Gallagher 2004) has been chosen here, and is used to learn, for a given domain, the best rates of application of variation operators.

Furthermore, another issue for DAE lies in the choice of the atoms that are used to describe each subgoal. Searching the space of complete states would result in a rapid explosion of the size of the search space. 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, previous experiments on different domains of temporal planning problems from the IPC benchmark series (Bibai, Schoenauer, and Savéant 2009) have demonstrated the need for a very careful choice of such atoms. The approach proposed here builds on the results of the racing procedure to promote few atoms that will be later used to construct sequences of partial goals for their evolutionary optimization.

Next section briefly introduces the *Divide-and-Evolve* planner and details the different components of the evolutionary algorithm, as well as the way it interacts with the embedded planner to compute the fitness of potential solutions. After having described the learning procedure, the last section presents and discusses preliminary results of DAE using this learning procedure on IPC benchmarks.

The Divide-and-Evolve Planner

In order to solve a planning problem $\mathcal{P}_D(I, G)$ (D for the domain), 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: representation of individuals (potential solutions), variation operators, and fitness.

Representation

An individual is a variable length list of subgoals, or partial states of the given problem. For the current experiment, the generation of partial states is based on the earliest time from which atoms can become true. Such **dates** can be estimated by a classical heuristic function (e.g h^1, h^2... (Haslum and Geffner 2000)). The set of all possibly atoms can be restricted to the atoms that are possibly true at a chosen date, and a partial state is built at each date by choosing randomly among atoms of the corresponding date. The sequence of

states is then built by preserving the chronology between atoms.

Although these restrictions can contain a large number of atoms, they can be reduced by choosing to keep only atoms built with a set of allowed predicates. We expect that this set can be learned by analyzing several optimized sequences of states given by the algorithm on several problems of the same domain.

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* (`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 two atoms are `mutex`. Nevertheless, it can be estimated with h^2 heuristic function (Haslum and Geffner 2000) in order to build mutex-free states.

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 one and the number of estimated start dates; For every chosen date, the number of atoms per state is chosen uniformly 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 uniformly independently chosen in both parents.

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 (**addStation**) or removing (**delStation**) a state; or at the state level by changing (**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. estimated mutual exclusion relations between atoms.

Applying Variation Operators Several parameters control the application of the variation operators. During an evolutionary run, two parents are chosen according to the selection procedure (see (Schoenauer, Savéant, and Vidal 2006; 2007)). With probability p_{cross}, they are recombined using the crossover operator. Each one then undergoes mutation with probability p_{mut}. When an individual must undergo mutation, one mutation operator amongst the 4 operators defined above is applied: four user-defined relative *weights* ($w_{addStation}, w_{delStation}, w_{addAtom}, w_{delAtom}$) are defined, and each operator has a probability proportional to its weight of being applied.

Fitness

The fitness of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling a independent '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 this work has chosen YAHSP (Vidal 2004), a lookahead strategy planning system for non-optimal STRIPS planning which uses the actions in the relaxed plan to compute reachable states in order to speed up the search process.

Here, there are two possibilities. If YAHSP succeeds in solving all $\mathcal{P}_D(S_k, S_{k+1})$, the makespan (or the number of actions or the total cost of actions) of the concatenation of the different plans (possibly after some compression step, see (Schoenauer, Savéant, and Vidal 2007) for detailed discussion) becomes the target objective of the fitness. In the case where YAHSP fails to solve one of the $\mathcal{P}_D(S_k, S_{k+1})$, the fitness promotes individuals which are "closer" to the goal, trying to ensure that individuals for which most subproblems are solved are later favoured by selection.

However, it rapidly became obvious that we needed to limit the exploration of the embedded planner in order to rapidly discard subproblems that could be even more difficult than the original problems. We have constrained YAHSP with a **maximal number of nodes** that it is allowed to use for any subproblem. First, when dealing with the initial population, a large number of nodes was allowed (e.g. 100000); then, the median of nodes that have been actually needed for YAHSP to solve the problems from the initial population is set as the limit for all subsequent calls to YAHSP - a 'rule of thumb' that appeared robust after some intensive preliminary experiments.

Parameter Learning

Some of the parameters of the algorithms have been definitely fixed after the early experiments reported in previous works (Schoenauer, Savéant, and Vidal 2006; 2007) (evolution strategy and stopping criteria). The present two-steps learning procecedure thus only involves choosing the probability and weights of each of the variation operators being used (the best domain-dependent search strategy), and then choosing which predicates will be used to describe the intermediate goals (the representation domain-dependent knowledge.

The Best Domain-Dependent Search Strategy

The naive way of tuning the parameters of evolutionary algorithms in order to solve a class of problems is to to use e.g. a full factorial Design Of Experiment over all problems, running a number of runs for each parameter configuration, and using some ANOVA statistical test to extract the significantly best parameter configurations. Originally proposed for solving the model selection problem in Machine Learning (Maron and Moore 1994), racing technique was introduced in Evolutionary context (Birattari et al. 2002) 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 continue 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 there is just one parameter configuration left, or the total number of experiments has been run.

The efficiency of such technique totally depends on the statistical test used for the comparison. Because no assumption can be made about the distribution of the results (e.g. normality), we have chosen here to use the nonparametric Friedman's two-way analysis of variances by ranks, as advocated in (Birattari et al. 2002; Yuan and Gallagher 2004; 2007).

Moreover, whatever statistical test, the user must set the number of initial runs before starting the comparison, and the confidence level. Here, 11 runs (the lowest number for significance of the statitical test) are run before starting the comparison for each problem tested during the learning and parameter configuration, and a confidence level of 0.025 (strong constraint for the acceptation of equality hypothesis between two parameter configurations) is used to select the best set of parameters in terms of the lowest number of actions and the lowest execution time. The racing process was stopped after at most 50 runs.

Importantly, and in order to reduce the overall executino time of the racing procedure, we assume some homogeneity amongst the instances of a given domain, and only perform racing using a small subset of all available instance, later extrapolating the results to the whole domain. These instances were selected by using YAHSP: First, YAHSP is launched on each available instances (e.g. the 'bootstrap' instances in IPC09 'learning track' competition) with a very limited time (e.g. 10 minutes). Two instances are then chosen, the one with the longest execution time amongst the ones that have been solved (unless all problems are unsolved), and the one with the lowest memory use amongst the unsolved ones.

Finally, rather than starting with a full factorial DOE, only twenty sets of parameter configurations were proposed to the racing procedure, also chosen after previous early experiments.

The Set of Predicates

At the end of the racing process, the set of predicates that seem useful for the domain at hand is chosen from the reuslts obtained by the best parameter configuration : the predicates that appear in at least 50% of all best solutions found by DAE on 11 first instances of the domain are later used to represent the intermediate states of all individuals. Note that we could also have chosen predicates according to the proportion (e.g. more than 20%) of their occurrence with regards to all the atoms contained in the solutions found by DAE.

Detailed DAE Results

Divide-and-Evolve has been implemented within the Evolving Objects framework[1], an open source, template-based, ANSI C++-STL-compliant evolutionary computation library. In order to illustrate here the solution quality of DAE results with the optimal results found by CPT (Vidal and Geffner 2006) and the best result found by LPG (Gerevini, Saetti, and Serina 2003), we used the `rovers` simple time domain of the third International Planning Competition (IPC), the `peg solitaire` domain of the sixth IPC sequential satisficing track, and `gold-miner` domain of the sixth IPC learning track for the preliminary tests.

The fixed **evolution engine** is 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** was 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.

After the racing, the DAE planner is run with the best parameter configuration found on each problem of all domains in at most 15min of CPU time.

Figures 1, 2 and 3 show for all algorithms, the makespan (or the number of actions or the total cost of actions) of all instances of each domain, each column corresponding to an instance (number on the X axis). For the deterministic YAHSP and CPT (or stochastic LPG), symbols ('@' and '#' respectively) indicate the makespan (resp. the number of actions or the total cost of actions) found. For the stochastic DAE, standard boxplots sketch the distribution of the 11 makespans.

Significantly, after the racing and the choice of predicates (with the distribution frequency), the solutions found by DAE are very close to the optimal solution found by CPT (Figures 2 and 3) or to the best value found by LPG (Figure 1). Solution plans found by allowing the predicates for which the distribution frequency is greater than 50% are on average better than those using more than 20% of proportions of predicate occurrences.

The other observation concerns the quality of the results, the choice of predicates using the distribution frequency often reduces the variance of the solution plans found by DAE. However, these choices can also increase the variance of the solution plans found by DAE. Nevertheless, the racing and predicates selection (with the distribution frequency) improve more often the average of results found by DAE on each problem.

Unfortunately, there has been cases where DAE with racing only found some optimal solutions, whereas DAE with racing and predicates selection could not. A poor choice of the allowed set of predicates could explain this behavior, and this would lead us to reconsider the hyothesis of homogeneiry of instances accross a given domain: this is the topic of on-going work.

Discussion and Conclusion

It is well known that parameter tuning is one of the weaknesses of evolutionary algorithms in general – and *Divide-and-Evolve* is not an exception. In this paper, we introduced a two steps learning approach in order to enhance DAE performance on a specific domain. Preliminary results on three

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

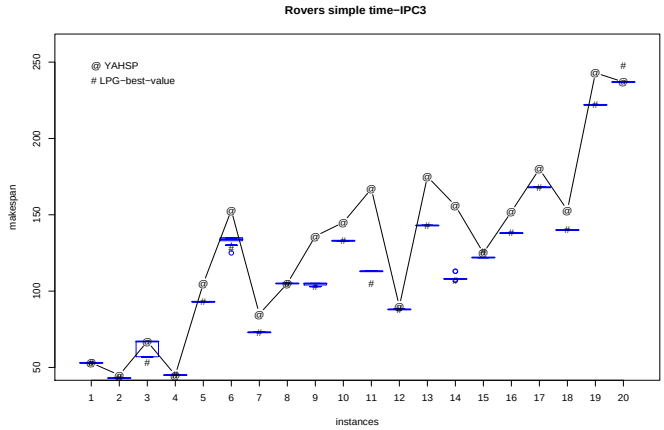

Figure 1: Best makespan found by LPG (#), YAHSP value (@) and DAE (standard boxplots sketch the distribution of the 11 makespans after the racing step and the predicate selection (more than 50% of frequency distribution of all solutions) on the `rovers simple time` IPC-3 domain.

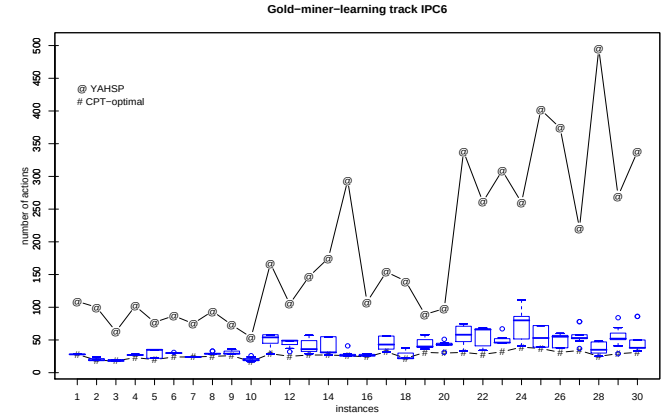

Figure 2: Optimal number of actions for CPT (#), YAHSP value (@) and DAE (standard boxplots sketch the distribution of the 11 makespans after the racing step and the predicate selection (more than 50% of frequency distribution of all solutions)) on the `gold-miner` IPC-6 domain.

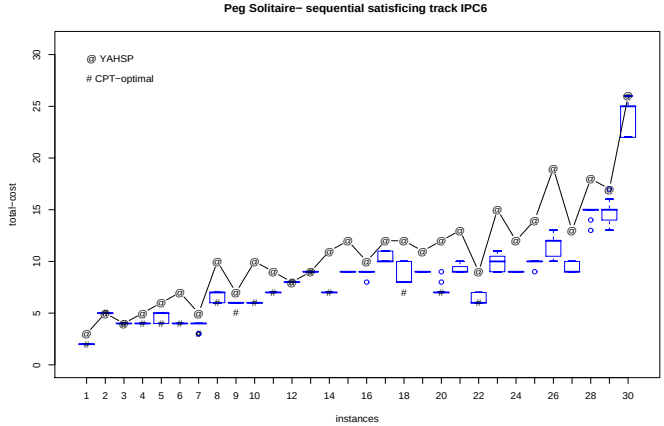

Figure 3: Optimal total cost of actions for CPT (#), YAHSP value (@) and DAE (standard boxplots sketch the distribution of the 11 makespans after the racing step and the predicate selection (more than 50% of frequency distribution of all solutions)) on the `peg solitaire sequential satisficing` IPC-6 domain.

IPC domains showed that our approach improves on average the quality of solutions obtained by DAE, though not uniformly over all tested domains.

In any case, there is still room for improvement in tuning DAE's parameters. First, the choice of the set of parameter configurations for the racing step is still an open issue. Although we obtained good results with twenty parameter configurations, we might miss a parameter configuration that would improve these results. An alternative to racing would be SPO (Bartz-Beielstein, Lasarczyk, and Preuss 2005), even though it only applies to numerical parameters.

Furthermore, the restriction of allowed predicates in the second step of our learning approach is still an open problem. We now plan to combine the frequency distribution with the occurrence proportions of predicates in order to choose more efficiently the most relevant set of allowed predicates of a specific domain.

References

Bartz-Beielstein, T.; Lasarczyk, C.; and Preuss, M. 2005. Sequential Parameter Optimization. In *Proc. CEC'05*, 773– 780. IEEE Press.

Bibai, J.; Schoenauer, M.; and Savéant, P. 2009. Divide-And-Evolve Facing State-of-the-art Temporal Planners during the 6^{th} International Planning Competition. In Cotta, C., and Cowling, P., eds., *Proc. EvoCOP'09*, LNCS 5482, 133–144. Springer-Verlag.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A racing algorithm for configuring metaheuristics. In *GECCO '02*, 11–18. Morgan Kaufmann Publishers Inc.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. On Managing Temporal Information for Handling Durative Actions in LPG. In *AI*IA 2003: Advances in Artificial Intelligence*. Springer Verlag.

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

Maron, O., and Moore, A. W. 1994. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In *In Advances in neural information processing systems 6*, 59–66. Morgan Kaufmann.

Schoenauer, M.; Savéant, P.; and Vidal, V. 2006. Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In Gottlieb, J., and Raidl, G., eds., *Proc. EvoCOP'06*. Springer Verlag.

Schoenauer, M.; Savéant, P.; and Vidal, V. 2007. Divide-and-Evolve: a Sequential Hybridization Strategy using Evolutionary Algorithms. In Michalewicz, Z., and Siarry, P., eds., *Advances in Metaheuristics for Hard Optimization*, 179–198. Springer.

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

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

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

Yuan, B., and Gallagher, M. 2007. Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In *Parameter Setting in Evolutionary Algorithms*, 121–142. Springer-Verlag.