Learning Divide-and-Evolve Parameter Configurations with Racing

Jacques Bibai, Pierre Savéant, Marc Schoenauer, Vincent Vidal

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 DAE planner implements the stochastic approach for domain-independent planning decomposition introduced in (Schoenauer, Savéant, and Vidal 2006; 2007). This planner optimizes either the makespan or the number of actions by generating ordered sequences of intermediate goals via a process of artificial evolution. The evolutionary part of DAE uses the Evolving Objects (EO) library, and the embedded planner it is based on 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 *Temporal Planning Problems* (TPPs), originally introduced in (Schoenauer, Savéant, and Vidal 2006; 2007). It uses an evolutionary algorithm to evolve an ordered sequence of subgoals; the resulting TPPs (going from one subgoal to the next) is then passed on to an embedded planner; if all those TPPs are solved, the concatenation of all corresponding subplans (after some compression step) is a solution of the initial TPP. The makespan (or number 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 computation. 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.

However, another issue for DAE lies in the choice of the atoms that are used to describe each subgoal: the goal of a TPP can be (and usually is) described only partially, i.e. by requiring only the value of a small number of atoms (instanciated predicates from the domain). Furthermore, 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 TPP from the IPC benchmark series (Bibai, Schoenauer, and Savéant 2009) has demonstrated the need for a very careful choice of such atoms. The approach proposed here builds on the results of the racing procedure to select a 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 the IPC-6 learning 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, variation operators, and fitness.

Representation

An individual, possible solution of the TPP at hand, is a (variable length) list of subgoals, or partial states of the given domain. In STRIPS representation model (Fikes and Nilsson 1971), a state is a list of boolean atoms. However, searching the space of complete states would result in a very fast combinatorial explosion of the search space size. Moreover, goals of TPP need only to be defined as partial states. It thus seemed practical to search only sequences of partial states. However, this raises the issue of the **choice of the**

atoms to be used to represent individuals, among all possible atoms.

The result of the experiments in many kinds of TPP on the IPC benchmarks (Bibai, Schoenauer, and Savéant 2009) 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 date can be estimated by a classical heuristic function (e.g $h^1, h^2...$ (Haslum and Geffner 2000)). The dates from which all atoms become true can then be discretized into the number of restrictions of the set of all possibly atoms, and a partial state is built at each date by choosing randomly among several atoms that are possible true at this date. The sequence of states is then built by preserving the estimated 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 choice of random atoms can lead to inconsistent partial states, because some sets of atoms can be *mutually exclusive* (`mutex` in short). Whereas it could be possible to allow `mutex` atoms in the partial states generated by DAE, and to let evolution discard them, it seems more efficient to a priori forbid them, as much as possible. 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 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, a 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 estimated 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. 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, four additional user-defined relative *weights* ($w_{addStation}, w_{delStation}, w_{addAtom}, w_{delAtom}$) are used to choose among the four mutation operators defined above: each operator has a probability proportional to its weight of being applied. At most one mutation operator is thus applied to each individual.

Fitness

The fitness of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling a 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 here, in this paper DAE uses 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.

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, and becomes the initial state of the next problem: initial states need to be completed (and denoting each problem as $\mathcal{P}_D(S_k, S_{k+1})$ is indeed an abusive notation). If all problems, $\mathcal{P}_D(S_k, S_{k+1})$ are solved by the chosen embedded planner, the individual is called *feasible*, and the concatenation of all solutions plans for all $\mathcal{P}_D(S_k, S_{k+1})$ is a global solution plan for $\mathcal{P}_D(S_0 = I, S_{n+1} = G)$. However, this plan can in general be optimised by parallelising some of its actions, in a step call *compression* (see (Schoenauer, Savéant, and Vidal 2007) for detailed discussion). The fitness of a feasible individual is the makespan (or the number of actions) of the compressed plan.

However, when 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 complete initial state is in fact unknown, and no makespan can be given to that individual. All such plans receive a fixed penalty cost such that the fitness of any infeasible individual is higher than that of any feasible individual. In order to nevertheless give some selection pressure toward feasible individuals, the relative rank of the first problem that the chosen embedded planner fails to solve is added to the fixed penalty, so infeasible individuals which solve the more subproblems are favoured by selection.

Finally, because the initial population contains randomly generated individuals, some of them might contain some subproblems that are in fact more difficult than the original global problems. It was necessary to limit the chosen embedded planner by adding some constraints in order to discard those subproblems. 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 YAHSP with a **maximal number of nodes** that it is allowed to use to solve any of the subproblems. We fixed the maximum number of nodes in two steps:

first to solve the initial population (at the initialization) we allowed a large number of node (e.g. 100000); and in the second step, for the rest of the algorithm, we allowed the median of nodes used in the solutions found at the initialization.

Parameter Learning

Because there are too many parameters to tune, some of them were fixed after preliminary experiments reported in the previous work (Schoenauer, Savéant, and Vidal 2006; 2007), i.e. evolution strategy and stopping criteria. So the two steps learning process only involves choosing the probability and weights of each of the variation operators being used (the best domain-dependent search strategy) and choosing predicates (the representation domain-dependent knowledge) for the intermediate goals. It first finds the best domain-dependent search strategy and then for this best domain-dependent search strategy finds the best set of allowed predicates.

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 try exhaustively all the parameter configurations over problems and, by means of statistical analysis over the results, the best configuration is extracted. The problem of doing so is that a lot of computational processing time might be thrown away while extensively evaluating very bad candidates. Originally proposed for solving the model selection problem in Machine Learning (Maron and Moore 1994), racing technique was introduced (Birattari et al. 2002) in order to focus the search in the most performing parameter configurations.

The general idea is that, while performing all runs on each parameter configuration, as soon as there is enough statiticals evidence that the current parameter configuration is worst than the best parameter configuration found so far, there is no reason to keep performing experiments with such configuration and so it can be discarded. Such cycle execution-comparison-elimination is repeated until there is just one parameter configuration left or if the maximum number of allowed experiments has been reached.

However, the efficiency of such technique totally depends on the selection of the statistical test to be applied in the comparison. Because no assumption can be made about the distribution of the results (e.g. normality), we have chosen to use the nonparametric Friedman's two-way analysis of variances by ranks. The application of racing using the Friedman's test, the so-called F-RACE, was deeply examined in (Birattari et al. 2002; Yuan and Gallagher 2004; 2007).

Moreover, two parameters are inherent to whatever statistical test chosen: the number of initial runs before starting the comparison; and the confidence level. In order to choose the best parameter configuration, we make 11 runs (the lowest significant number for the statitical test) before starting the comparison for each problem tested during the learning and parameter configuration. We used 0.025 confidence level (strong constraint for the acceptation of equality hypothesis between two parameter configurations) to select the best set of parameters in terms of the lowest number of actions and the lowest execution time.

Because the test of a parameter configuration can take too much time, we decided to choose only some problems for the racing. These problems were selected by using YAHSP: we tried to solve each bootstrap (example) problem with YAHSP with a limited time (e.g. 10 minutes). After this step we selected two problems for the learning procedure, one of which has been solved by YAHSP (unless all problems are unsolved). The solved problem chosen was that with the longest execution time; the unsolved problem was that with the lowest memory.

In order to reduce the space of the variation operator parameters we selected twenty sets of parameter configurations after preliminary experiments. The racing process was stopped after at most 50 runs.

The Set of Allowed Predicates

At the end of the racing process we analyze all the results obtained by the best parameter configuration to derive a set of useful predicates of the domain. We chose to keep the predicates that appear in at least 50% of all best solutions found by DAE on 11 first instances. However, we could also choose predicates according to the proportion (e.g. more than 20%) of their occurrence with regard 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++-compliant evolutionary computation library. In order to compare the solution quality of DAE with the optimal results found by CPT, we used the gold-miner domain of the sixth International Planning Competition (IPC6) learning track for the preliminary tests.

The chosen **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 target problem of the gold-miner domain in at most 15min of CPU time.

Figures 1, 3 and 6 show for all 3 algorithms, the makespan of all target instances of the gold-miner domain, each column corresponding to an instance (number on the X axis). For the deterministic YAHSP and CPT, symbols ('@' and '#' respectively) indicate the makespan found. For the stochastic DAE, standard boxplots sketch the distribution of the 11 makespans.

Figure 2 shows, for all results found by DAE, the distribution of predicate frequency (i.e. the number of solution

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

plans containing at least one occurrence of a given predicate divided by the total number of solution plans).

Figure 5 shows the occurence proportion of a predicate (i.e. the number of occurrences of a given predicate divided by the total number of atoms).

Significantly, after the racing, the solutions found by DAE are very close to the optimal solution found by CPT (see 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 (see Figure 3 and 6). We note that the distribution frequency of a predicate is not correlated to its occurence proportion (see Figures 2 and 5).

The other observation concerns the quality of the results (see Figure 4), for problem 27 for instance, the choice of predicates (with the distribution frequency) often reduces the variance of the solutions plans found by DAE. However, for the problem 11, these choices can also increase the variance of the solutions plans found by DAE. Nevertheless, for the problem 30, the racing and predicates selection (with the distribution frequency) improve more often the average of results found by DAE on each problem.

We can also notice that at least one run of DAE with racing and predicates selection found 13 times the optimal value when DAE after the racing has found just 7 (see Figure 4). However, DAE racing only found some optima solutions whereas DAE with racing and predicates selection cannot. The choice of the allowed set of predicates could explain this behavior. Indeed, a set of predicates can be relevant for one problem and not for another one.

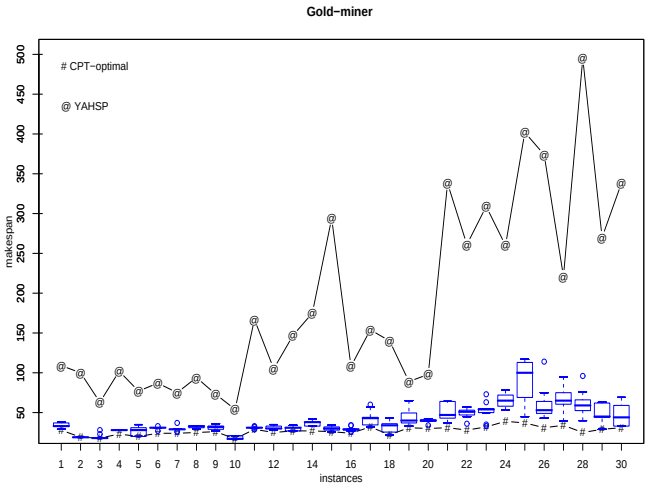

Figure 1: Optimal number of actions for CPT (#), YAHSP value (@) and DAE (standard boxplots sketch the distribution of the 11 makespans after the racing step) on the `gold-miner` IPC-6 domain.

Discussion and Conclusion

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

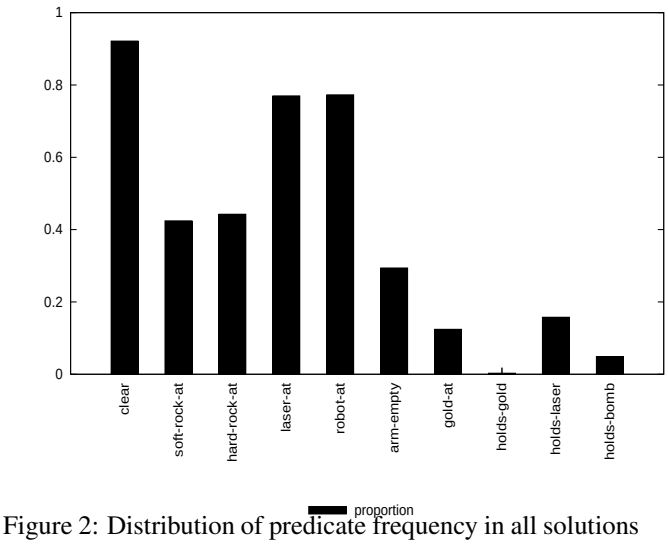

Figure 2: Distribution of predicate frequency in all solutions plans found by DAE for 11 executions of each target problem of the `gold-miner` IPC-6 domain.

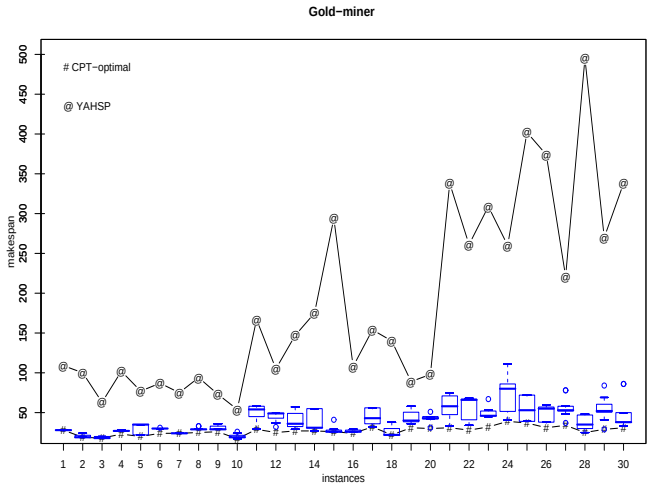

Figure 3: Optimal number of actions for CPT (#), YAHSP value (@) and DAE (standard boxplots sketch the distribution of the 11 makespans after the racing step and the predicate selection (more than 50% of frequency distribution of all solutions - see Figure 2)) on the `gold-miner` IPC-6 domain.

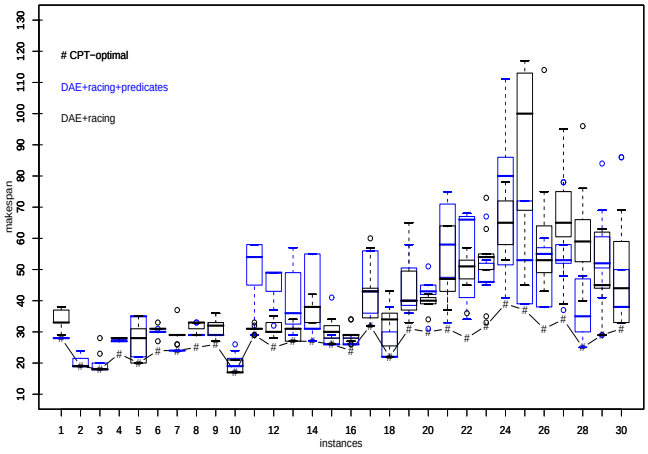

Figure 4: Comparison between DAE after the racing step and the predicate selection (blue boxplots) (more than 50% of frequency distribution of all solutions - see Figure 2)) and DAE after the racing step (black boxplots) on the `gold-miner` IPC-6 domain. Optimal number of actions for CPT (#).

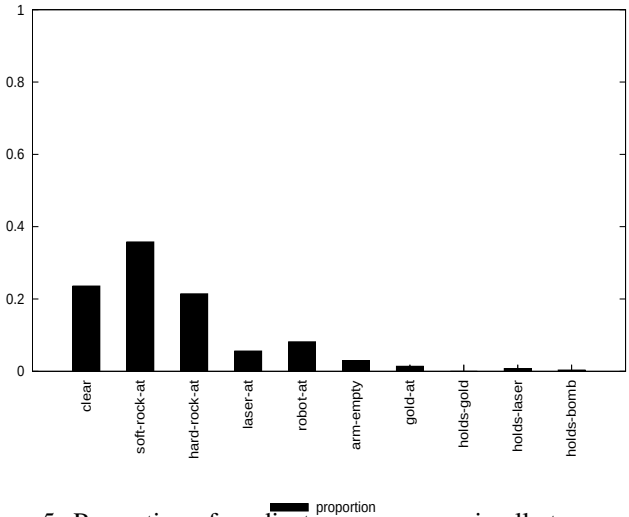

Figure 5: Proportion of predicates occurrences in all atoms appearing in all solutions plans found by DAE for 11 executions of each target problem of the `gold-miner` IPC-6 domain.

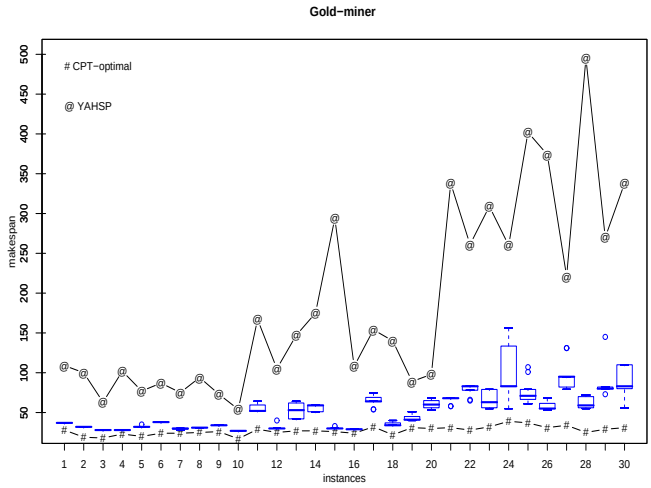

Figure 6: Optimal number of actions for CPT (#), YAHSP value (@) and DAE (standard boxplots sketch the distribution of the 11 makespans after the racing step and predicate selection (more than 20% of occurences proportion- see Figure 5)) on the `gold-miner` IPC-6 domain.

IPC-6 gold-miner domain showed that our approach can improve the average quality of solutions obtained by DAE after the racing step.

But 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.

According to the results, the selection of allowed predicates for the second step of our learning approach is still open. We 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. These directions will be pursued during further research.

References

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., *Ninth European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2009)*, number 5482 in Lecture Notes in Computer Science, 133–144. Springer-Verlag.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A racing algorithm for configuring metaheuristics. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, 11–18. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

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

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. 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.