Divide-and-Evolve: the Marriage of Descartes and Darwin

Johann Dreo **Pierre Savéant**
Thales Research & Technology
Palaiseau, France
first.last@thalesgroup.com

Marc Schoenauer
INRIA Saclay & LRI
Orsay, France
marc.schoenauer@inria.fr

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

Abstract

DAE$_{\text{X}}$, the concrete implementation of the *Divide-and-Evolve* paradigm, is a domain-independent satisficing planning system based on Evolutionary Computation. The basic principle is to carry out a *Divide-and-Conquer* strategy driven by an evolutionary algorithm. The key components of DAE$_{\text{X}}$ are a state-based decomposition principle, an evolutionary algorithm to drive the optimization process, and an embedded planner X to solve the sub-problems. The release that has been submitted to the competition is DAE$_{\text{YAHSP}}$, the instantiation of DAE$_{\text{X}}$ with the heuristic forward search YAHSP planner. The marriage of DAE and YAHSP matches a clean role separation: YAHSP gets a few tries to find a solution quickly whereas DAE controls the optimization process.

Introduction

This section introduces the main principles of the satisficing planner DAE, referring to (Bibaï et al. 2010c) for a comprehensive presentation. DAE$_{\text{X}}$, the concrete implementation of the *Divide-and-Evolve* paradigm, is a domain-independent satisficing planning system based on Evolutionary Computation (Schoenauer, Savéant, and Vidal 2006). The basic principle is to carry out a *Divide-and-Conquer* strategy driven by an evolutionary algorithm. The algorithm is detailed in (Bibaï et al. 2010a) and compared with state-of-the-art planners.

Given a planning problem $P = \langle A, O, I, G \rangle$, where A denotes the set of atoms, O the set of actions, I the initial state, and G the goal state, DAE$_{\text{X}}$ searches the space of sequences of partial states $(s_i)_{i \in [0,n+1]}$, with $s_0 = I$ and $s_{n+1} = G$: DAE$_{\text{X}}$ looks for the sequence such that the plan σ obtained by compressing subplans σ_i found by some embedded planner X as solutions of $P_i = \langle A, O, \hat{s}_i, s_{i+1} \rangle_{i \in [0,n]}$ has the best possible quality (with $\hat{s_i}$ denoting the final state reached by applying σ_{i-1} from $\hat{s}_{i-1}$). Each intermediate state $(s_i)_{i \in [1,n]}$ is first seen as a set of goals and then completed as a new initial state for the next step by simply applying the plan found to reach it. In order to reduce the number of atoms used to describe these states, DAE relies on the admissible heuristic function h^1 (Haslum and Geffner 2000a): only the ones that are possibly true according to h^1 are considered. A diagram of the decomposition approach in DAE is depicted on figure 1.

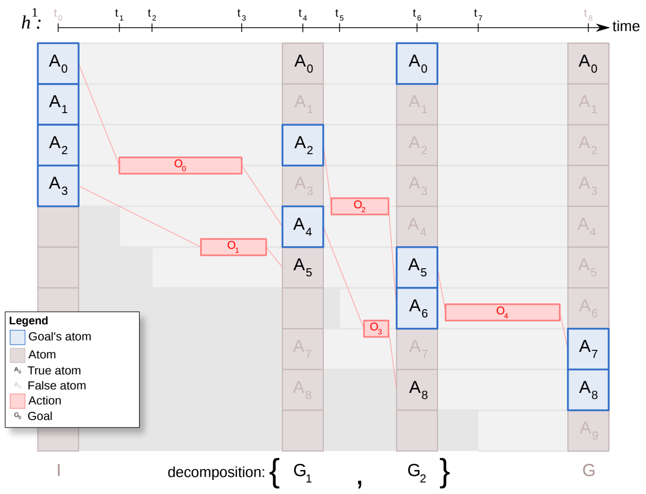

Figure 1: The decomposition approach used in DAE.

Furthermore, mutually exclusive atoms, which can be computed at low cost, are also forbidden in intermediate states s_i. These two rules are strictly imposed during the random initialization phase, and progressively relaxed during the search phase. The compression of subplans is required by temporal planning where actions can run concurrently: a simple concatenation would obviously not produce the minimal makespan.

Due to the weak structure of the search space (variable-length sequences of variable-length lists of atoms), Evolutionary Algorithms (EAs) have been chosen as the method of choice: EAs are metaheuristics that are flexible enough to explore such spaces, as long as they are provided with some stochastic *variation operators* (aka *move operators* in the heuristic search community) – and of course some objective function to optimize.

Variation operators in DAE are (i) a crossover operator, a straightforward adaptation of the standard one-point crossover to variable-length sequences; and (ii) different mutation operators, that modify the sequence at hand either at the sequence level, or at the state level, randomly adding or removing one item (state or atom).

The objective value is obtained by running the embedded planner on the successive subproblems. When the goal state is reached, a feasibility fitness is computed based on the compression of solution subplans, favoring quality; otherwise, an unfeasibility fitness is computed, implementing a gradient towards satisfiability (see (Bibaï et al. 2010c) for details).

DAE can embed any existing planner, and has to-date been successful with both the optimal planner CPT (Vidal and Geffner 2004) and the lookahead heuristic-based satisficing planner YAHSP (Vidal 2004). The latter has been demonstrated to outperform the former when used within DAE (Bibaï et al. 2010d), so only DAE$_{\text{YAHSP}}$ has been considered in this work.

The target is thus temporal satificing planning with conservative semantics, cost planning and classical STRIPS planning. The marriage of DAE and YAHSP matches a clean role separation: YAHSP gets a few tries to find a solution quickly whereas DAE controls the optimization process. In the current release we have introduced an initial estimation processing of the maximum number of tries allowed to YAHSP for all individual evaluations. This parameter is crucial for the time consumption of the algorithm.

Algorithms

DAE$_{\text{X}}$ is an evolutionary algorithm, which basically mimics a biological evolution as a stochastic process (i.e. using biased random search in an iterative manner). Figure 2 depicts the main components of the evolution engine of DAE$_{\text{YAHSP}}$.

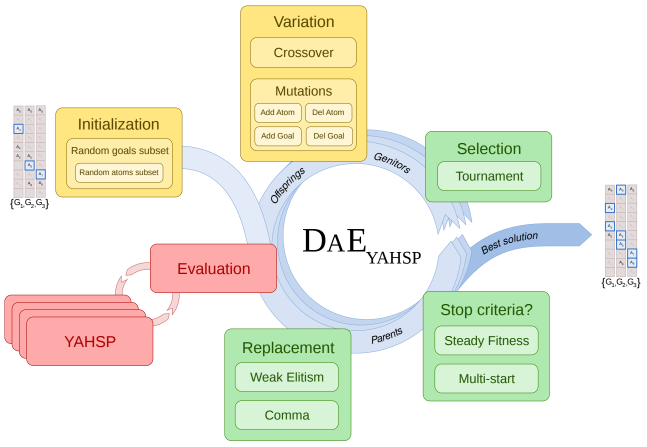

Figure 2: The evolution engine used in DAE$_{\text{YAHSP}}$. Yellow boxes indicates problem-dependent operators, green ones problem-independent operators and red boxes indicates the planner-dependent fitness evaluation. The output of the evolutionary algorithm is a decomposition of the problem.

The fitness implements a gradient towards feasibility for unfeasible individuals and a gradient towards optimality for feasible individuals. Feasible individuals are always preferred to unfeasible ones. Population initialization as well as variation operators are driven by the critical path h^1 heuristic (Haslum and Geffner 2000b) in order to discard inconsistent state orderings, and atom mutual exclusivity inference in order to discard inconsistent states. These two computations are done by YAHSP in an initial phase.

Beside a standard one-point crossover for variable length representations, four mutations have been defined: addition (resp. removal) of a goal in a sequence, addition (resp. removal) of an atom in a goal.

Variation operators relax the strictly h^1 ordering of atoms within individuals, since it is only a heuristic estimate.

The selection is a comparison-based deterministic tournament of size 5.

For the sequential release, Darwinian-related parameters of DAE$_{\text{X}}$ have been fixed after some early experiments (Schoenauer, Savéant, and Vidal 2006) whereas parameters related to the variation operators have been tuned using the Racing method (Bibaï et al. 2010b). It should be noted that, due to the conditions of the competition, the parameter setting is global to all domains. In (Bibaï et al. 2010b) we showed that a specific tuning for an instance provides better results as expected and that what we would do for a real-life planning task.

We added two novelties to the version described in (Bibaï et al. 2010a). One important parameter is the maximum number of expanded nodes allowed to the YAHSP subsolver which defines empirically what is considered as an easy problem for YAHSP. As a matter of fact, the minimum number of required nodes varies from few nodes to thousands depending of the planning task. In the current release this number is estimated during the population initialization stage. An incremental loop is performed until the ratio of feasible individuals is over a given threshold or a maximum boundary has been reached. By default this number is doubled at each iteration until at least one feasible individual is produced or 100,000 has been reached.

Furthermore we add the capability to perform restarts within a time contract in order to increase solution quality.

The fitness used for the competition differs from the one described in (Bibaï et al. 2010a). The fitness for bad individuals has been simplified by withdrawing the Hamming distance to the goal. The new fitness depends only on the "decomposition distance": the number of intermediate goals reached and more specifically the one that are "useful". A useful intermediate goal is a goal that require a non-empty plan to be reached.

Implementation

The implementation of DAE$_{\text{X}}$ has been made with the ParadisEO framework[1] which provides an abstract control structure to develop any kind of evolutionary algorithm in C++. YAHSP is written in the C language. The source code is available under an open-source license and the version used for the competition has the hash `9a46716` in the official repository[2].

In order to speed up search, a memoization mechanism has been introduced in YAHSP and carefully controlled to

[1]`http://paradiseo.gforge.inria.fr/`

[2]`https://gforge.inria.fr/git/paradiseo/paradiseo.git`

leave memory space for DAE. Indeed, most of the time during a run of YAHSP, and as a consequence during a run of $\text{DAE}_{\text{YAHSP}}$, is spent in computing the h^{add} heuristic for each encountered state (see (Vidal 2011) for more details about the algorithms of the new version of the YAHSP planner). During a single run of YAHSP, duplicate states are discarded; but during a run of $\text{DAE}_{\text{YAHSP}}$, the same state can be encountered multiple times. We therefore keep track of the h^{add} costs of all atoms in the problem for each state, in order to avoid recomputing these values each time a duplicate state is reached. This generally leads to a speedup comprised between 2 and 4. When $\text{DAE}_{\text{YAHSP}}$ runs out of memory, which obviously happens much faster with the memoization strategy, all stored states and associated costs are flushed. More sophisticated strategies may be implemented, e.g. flushing the oldest or less often encountered states; but we found that the simplest solution of completely freeing the memoized information was efficient enough.

Several biases have been introduced in YAHSP, in order to help $\text{DAE}_{\text{YAHSP}}$ finding better solutions. The main one is that actions of lower duration are preferred to break ties between several actions of same h^{add} cost, when computing relaxed plans and performing the relaxed plan repair strategy. Another bias is that the cost incrementation made during h^{add}, which is usually equal to 1 for each applied action, is made equal to either the duration or the cost of the action. Although these biases do not change a lot the quality of the plans produced by YAHSP alone, we found that they are of better help to $\text{DAE}_{\text{YAHSP}}$. However, introducing such biases is not very satisfactorily; it would be better to exactly use the version described in (Vidal 2011). We still have to better investigate the relationships between the evolutionary engine and the embedded planner, in order to determine how to manage such kind of biases and other tie-breaking strategies.

The version submitted to the sequential multi-core track use a parallelized evaluation operator that dispatch the fitness computation across multiple processes using message passing. No change is made to the $\text{DAE}_{\text{YAHSP}}$ algorithm, the implementation uses parallel operators wrappers available in the ParadisEO framework, with a static assignment of jobs. Note that while the source code permits a parallelization at the multi-starts level, it is not used in the competition.

Acknowledgments

This work is being partially funded by the French National Research Agency (ANR) through the COSINUS programme, under the research contract DESCARWIN (ANR-09-COSI-002).

References

Bibaï, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010a. An Evolutionary Metaheuristic Based on State Decomposition for Domain-Independent Satisficing Planning. In 20th *International Conference on Automated Planning and Scheduling (ICAPS-2010)*, 18–25. AAAI Press.

Bibaï, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010b. On the Generality of Parameter Tuning in Evolutionary Planning. In 20th *Genetic and Evolutionary Computation Conference (GECCO'10)*, 241–248. ACM Press.

Bibaï, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010c. An Evolutionary Metaheuristic Based on State Decomposition for Domain-Independent Satisficing Planning. In R. Brafman et al., ed., 20th *International Conference on Automated Planning and Scheduling (ICAPS-10)*, 18–25. AAAI Press.

Bibaï, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010d. On the Benefit of Sub-Optimality within the Divide-and-Evolve Scheme. In Cowling, P., and Merz, P., eds., *Proc.* 10th *EvoCOP*, 23–34. LNCS 6022, Springer Verlag.

Haslum, P., and Geffner, H. 2000a. Admissible Heuristics for Optimal Planning. In 5th *Int. Conf. on AI Planning and Scheduling (AIPS 2000)*, 140–149.

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

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., 6th *European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'06)*. Springer Verlag.

Vidal, V., and Geffner, H. 2004. Branching and Pruning: An Optimal Temporal POCL Planner Based on Constraint Programming. In *Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 570–577. AAAI Press.

Vidal, V. 2004. A Lookahead Strategy for Heuristic Search Planning. In 14th *International Conference on Planning and Scheduling (ICAPS-04)*, 150–159. AAAI Press.

Vidal, V. 2011. YAHSP2: Keep It Simple, Stupid. In 7th *International Planning Competition (IPC-2011), Deterministic Part.*