DAE: Planning as Artificial Evolution – (Deterministic part)

Jacques Bibai, Pierre Savéant, Marc Schoenauer, Vincent Vidal

DAE: Planning as Artificial Evolution (Deterministic part)

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 Futurs, 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). The purpose of this planner is to optimize the makespan, or the number of actions, by generating ordered sequences of intermediate goals via a process of artificial evolution. For the evolutionary part we used the Evolving Objects (EO) library, and to solve each intermediate subproblem we used the constraint-based optimal temporal planner CPT (Vidal and Geffner 2004; 2006). Therefore DAE can only solve problems that CPT can solve. Compression of subplans into a global solution plan is also achieved efficiently with CPT by exploiting causalities found so far. Because the selection of predicates for intermediate goal generation is still an open question, we have submitted two planners DAE1 and DAE2 that use different strategies for the generation of intermediate goals. An empirical formula has been defined to set a limit on the number of backtracks allowed for solving the intermediate subproblems.

Introduction

Divide-and-Evolve is an approach for Planning Decomposition originally introduced in (Schoenauer, Savéant, and Vidal 2006; 2007), and based on an evolutionary algorithm that searches the space of state decompositions. The *Divide-and-Evolve* principle consists of dividing the initial problem into subproblems by generating sequences of intermediate goals which define hopefully easier subproblems, and then rebuilding the global solution. The CPT temporal planner (Vidal and Geffner 2004; 2006) is used not only to solve the consecutive subproblems, but also to compress the concatenation of subplans so obtained, in order to take advantage of the potential concurrency of actions in temporal planning problems.

Divide-and-Evolve (DAE) has been implemented within the *Evolving Objects* framework (http://eodev.sourceforge.net), a template-based, ANSI-C++ compliant Evolutionary Computating Open Source library.

The Divide-and-Evolve engine

Evolutionary algorithms are stochastic optimization algorithms which mimics the natural evolution loop to find solutions. The principle is to sample the hilly search landscape by following a trade-off strategy between diversity of exploration and exploitation of local optima. In the metaphor, selection/replacement operators promote the best individuals (i.e. the candidate solutions) whereas crossover and mutation operators perform random variations. Adaptation of the individual to the environment (i.e. quality of candidate solutions) is captured by the fitness function.

Figure 1 describes the main DAE generation loop. More details on the representation, the compression process, the variation operators and the *blind initialization* implemented here, can be found in the paper entitled *Evolutionary Planning Decomposition* and submitted to ICAPS 2008. Termination of the algorithm is defined by a limit on the number of generations without any improvement or by a time limit.

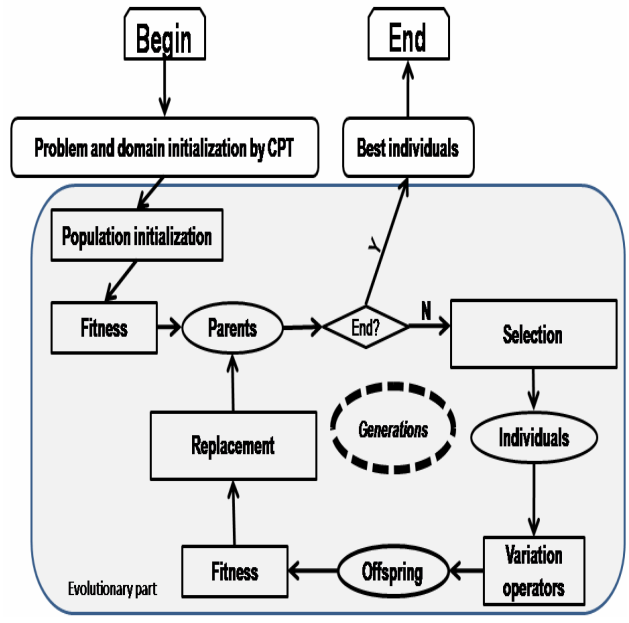

Figure 1: The main DAE generation loop

Representation

Individuals are represented as variable length lists of partial states. The initial and goal state, will not be modified by evolution, and hence are not encoded in the individual.

States contain atoms built with predicates of arity either 1 or 2. Atom selection for state generation, either at the initialization stage or within variation operators, is predicate-oriented.

Fitness

The fitness or objective function of an individual decides whether or not it will be selected in the next generation. Here, we have two kinds of behavior. Assuming that the local planner can solve sequentially all intermediate states of an individual, the total makespan of the global plan becomes the target objective of the fitness. The other behavior concerns individuals for which a failure has occurred (CPT fails to solve it). In this case the fitness promotes individuals which are "closer" to the goal. More formally, the fitness is defined by the following algorithm:

$k \leftarrow 0$
$w \leftarrow initialState()$
$s \leftarrow \emptyset$
while $s \neq Goal$
  $s \leftarrow nextIntermediateGoal()$
  $sol_k \leftarrow solve(w, s)$
  if $sol_k = fail$ **then**
    return $\alpha * (n * (10 * (g - s) + 11) - u)$
    else
      $w \leftarrow planExec(w, sol_k)$
      $k \leftarrow k + 1$
endWhile
$sol_{global} \leftarrow compress_{0 \leq j \leq k}(sol_j)$
return $m + (n - u + 1)/m + uBks/mBks$

Where m is the total makespan of the compressed plan, n the number of intermediate states, $uBks$ the number of used backtracks, $mBks$ the maximum number of authorized backtracks, $\alpha = 10^{60}$ a constant (feasible individual is always better than unfeasible one), g the number of goal atoms in the state where the failure has occurred, u the number of *useful* intermediate states (makespan > 0) and s the number of subgoals achieved before the failure.

Variation operators

Variation operators are classified depending on the number of individual they use to create a new individual. Those that need two parents are called crossovers and the others (that use only one) are called mutations.

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 (**changeAtom**) or removing (**delAtom**) some atoms in the given state.

Note that the initialization process and these variation operators maintain a local consistency, i.e. mutual exclusion relations between conflicting actions (mutex) in order to build mutex-free states.

Parameters

The evolution engine has been chosen to be a (10+70)-ES: 10 parents generate 70 offspring (no selection at this point), and the best of those 80 individuals become the parents of the next generation.

The number of states in an individual during initialization is uniformly chosen between 1 and the number of atoms in the goal of the problem, divided by the number of atoms per state; the number of atoms per state is chosen in $[1, 3]$.

After the initialization phase, CPT provides statistics that we use in an empirical formula to set a limit on the number of backtracks allowed for solving each subproblem. This formula can be seen as a mean of the number of causals links plus the number of actions relative to the number of atoms generated after actions grounding and the number of nodes relative to the number of conflicts. Several experiments have shown that we need more backtracks to solve the last subproblem. In this case we define another formula.

$$bks = \begin{cases} bks_{state} = \#Ga * (\frac{\#nodes}{\#conflicts} + 2 * (\frac{\#causals + \#actions}{\#atoms})) \\ \\ bks_{goal} = 7 * bks_{state} \end{cases} \quad (1)$$

where $\#Ga$ is the number of goal atoms, $\#causals$ the number of causal links, $\#actions$ the number of actions and $\#atoms$ the number of atoms generated after action grounding.

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, 4 additional user-defined relative *weights* ($w_{addStation}$, $w_{delStation}$, $w_{changeAtom}$, $w_{delAtom}$) are used. In order to choose among the 4 mutation operators each operator has a probability proportional to its weight of being applied.

The setting of the probabilities of individual-level application of crossover and mutation (p_{cross} and p_{mut}) and the relative weights of the 4 mutation operators ($w_{addStation}$, $w_{delStation}$, $w_{changeAtom}$, $w_{delAtom}$) are : (0.3,0.8) for (p_{cross}, p_{mut}), and (35, 3, 35, 7) for ($w_{addStation}$, $w_{delStation}$, $w_{changeAtom}$, $w_{delAtom}$).

Deterministic planners

We will present two planners that use different strategies for the description of intermediate goals, because the selection of predicates is still an open question. The first one uses only the predicates that are present in the goal (DAE1) of the problem and the second one uses a subset of all predicates (DAE2). These predicates of this subset are chosen using the rule shown in figure 2.

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. The parameters used here have

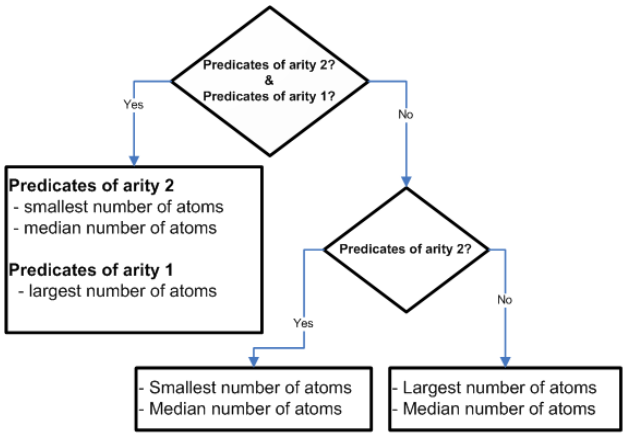

Figure 2: Rule for predicate choice

been validated on previous IPC benchmarks. Although these parameters are robust, there may be some domains for which the same parameters will not be adapted. Therefore we will present two other planners for the learning track of which the aim will be to find the best set of parameters for a specific domain.

References

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. 2004. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. In *Proceedings of AAAI-2004*, 570–577.

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