An Optimizing and Learning Algorithm Framework used for Parameter Tuning of the Divide-and-Evolve Method

Mátyás Brendel[1] **Johann Dréo** [2] **Pierre Savéant**[2] **Marc Schoenauer**[1] **Vincent Vidal**[3]

[1]Projet TAO
INRIA Saclay & LRI
Université Paris Sud
Orsay, France
matthias.brendel@lri.fr
marc.schoenauer@inria.fr

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

[3]ONERA – DCSD
Toulouse, France
vincent.vidal@onera.fr

Abstract

Abstract

Introduction

Parameter tuning is basically a general optimization problem applied off-line to find the best parameters for a complex algorithm, for example for Evolutionary Algorithms (EAs). Whereas the efficiency of EAs has been demonstrated on several application domains (see for instance (Yu et al. 2008) and (Lobo, Lima, and Michalewicz 2007)), they usually need computationally expensive parameter tuning. Consequently, one is tempted to use the 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 are similar to his one.

Being a general optimization problem, there might be as many parameter tuning algorithms as optimization techniques. Several methods have been proposed since Grefenstette's pioneering work (Grefenstette 1986). For a good comparison see **??**. Though we have some experience both with Racing (**??**), REVAC (**??**) and ParamILS ((Hutter et al. 2009)), one faces always the same generalization issue: can a parameter set that has been optimised for a given problem be successfully used to some other problem? The answer of course depends on the similarity of the two problems. However, even in an optimization domain (like AI Planning) precisely defined, there are very few examples today of meaningful similarity measures between optimization problems. Moreover, until now, sufficiently precise and accurate features have not been specified that would allow the user to describe the problem, so that the optimal parameter set could be learned from this features, and carried on to other problems with similar description. No design of a general learning framework is known to us and no general experiments has been carried out yet - at least to our knowledge - with some representative domains of AI planning.

The one exception we are aware of is the work (Hutter et al. 2006) in the SAT domain, in which many relevant features have been gathered based on half a century of SAT-research, and hundreds of papers. Extensive parameter tuning on several thousands of instances have allowed the authors to learn a meaningful mapping between between the features and parameters to running time, using function regression. Optimizing this model consisting of basis function makes it possible to choose the optimal parameters. This paper aims to generalize this work made in the SAT domain, we, however aim not to optimize running time, but fitness value, which is not a difference causing any problem. We design the aforementioned framework for any domain, and start with some experimenting. The machine learning techniques we use will also be different.

An AI planning task written in PDDL (more details in Section) 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 (Hutter et al. 2006). In this paper however, we make a big step on the framework for parameter tuning applied generally for each domain in AI with a preliminary set of features. Our Learn and Optimize (LaO) framework consists of the combination of optimizing (i.e. parameter tuning) and of learning the mapping between features and best parameters. We benefit from learning already in the op-

timization phase using the learned model as in other model based optimization techniques. Or LaO framework has the capability in theory to solve intra-domain and inter-domain generalization.

LaO can of course be applied to any other algorithm, but in this paper we are considering our special, an Evolution Strategy (ES) algorithm, called Divide and Evolve (DaE), which is an EA applied to solve AI problems. We take the DaE as a black-box algorithm and did not modify it in this paper. The realization described in (Jacques Bibai et al. 2010b) is used.

The paper is organized as follows: AI Planning Problems and the basic YASHP Solver is briefly introduced in Section . Section describes our intermediate, Divide and Evolve Method. Section describes our new, top level method: Learn and Optimize, used for parameter tuning.

AI Planning Problems and the YASHP Solver

An Artificial Intelligence (AI) planning task is defined by the triplet of an initial state, a goal state, and a set of possible actions. An action modifies the current state and can only be applied if certain conditions are met. A solution to a planning task is an ordered list of actions, whose execution from the initial state achieves the goal state. The quality criterion of a plan depends on the type of available actions: in the simplest case (e.g. STRIPS domain) it is number of actions; it may be the total cost of the actions with cost; it may also be the total makespan for so called durative actions.

Domain-independent planners rely on the Planning Domain Definition Language PDDL2.1 (Fox and Long 2003). 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.

The description of a planning consists of two separate parts (files): the generic domain on the one hand and a specific instance scenario on the other hand. The domain file specifies object types, predicates and actions which define 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 followed by a list of object identifiers: (PREDICATE_NAME OBJ_1 ... OBJ_N).

The initial state is complete, whereas the goal might be a partial state. 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 in the following as $\mathcal{P}_D(I, G)$.

Our Divide and Evolve Algorithm

Our Divide and Evolve (DaE) approach is fully described in (Jacques Bibai et al. 2010b), and no change has been made in the algorithm used in this paper. We consider DaE here as a black-box algorithm, whose parameter has to be tuned. In this section nevertheless we present a shorter description of DaE.

Early approaches to AI Planning using Evolutionary Algorithms directly handled possible solutions, i.e. possible plans: an individual is an ordered sequence of actions see (Spector 1994), (Muslea 1997), (Westerberg and Levine 2000), (Westerberg and Levine 2001), and (Brié and Morignot 2005). However, as it is often the case in Evolutionary Combinatorial optimizamtion, 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 (Hart, Krasnogor, and Smith 2005). Along those lines, though relying on an original "memetization" principle, a novel hybridization of Evolutionary Algorithms (EAs) with AI Planning, termed Divide-and-Evolve (DaE) has been proposed (Schoenauer, Savéant, and Vidal 2006) (Schoenauer, Savéant, and Vidal 2007). For a complete formal description, see (Jacques Bibai et al. 2010a).

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.

The fitness (quality criterion) 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 could be used, but since guaranty of optimality at

all calls is not mandatory in order for DaE to obtain good quality results, a sub-optimal planner, YAHSP is used. YAHSP2 (Vidal 2004) is a lookahead strategy planning system for sub-optimal planning which uses the actions in the relaxed plan to compute reachable states in order to speed up the search process.

We will now describe its main components: the problem-specific representation of individuals, fitness, and variation operators.

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 (Bibai, Savéant, and Schoenauer 2009) 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... (Haslum and Geffner 2000)). 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 (Bibai et al. 2008). 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.

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 (Haslum and Geffner 2000) 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.

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 1 and the number of estimated start times. 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 any other atom that is already there.

One-point crossover is used, adapted to variable-length representation in that both crossover points are independently chosen, uniformly in both parents.

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.

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 at both 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.

Note that the initialization process and these variation operators maintain the chronology between atoms in a

[1] Several atoms are mutually exclusive when there exists no plan that, when applied to the initial state, yields a state containing them all.

sequence of states and the local consistency of a state, i.e. avoiding pairwise mutexes.

The Learn and Optimize Parameter Tuning Framework

The General LaO Framework

As already mentioned, parameter tuning is actually a general global optimization problem, thus the main issue of local optimality consists also a problem here. But a further problem arises, when parameter tuning is applied for AI planning, and this is the generality of the tuned parameters. Parameter tuning for one AI instance has some meaning in the sense that a better solution to that instance can be achieved. However, as (Bibai et al. 2010) shows, this is costly. On the other hand, this very same paper also shows that individually tuned parameters may outperform the parameters tuned for the whole domain. Tuning parameters for a whole domain is a magnitude more costly, since a representative set of instances have to be evaluated. There seems to be no good solution.

If the parameters are generalized for another instance in the same domain (intra-domain generalization), the problem is that there are instances with very different complexity in the same domain, simply, because the size of the problem. The same issue arises if one wants to carry out parameter tuning for the whole domain, i.e. by testing with a representative set of instances of that domain. Since the optimum values of the parameters might change from instance to instance, only a "dull" average-like setting may be computed.

If the parameter-setting is generalized for an instance in another domain (extra-domain generalization) the same problems may arise, but furthermore, also differences between the domains may cause a problem. Even an instance of similar complexity may require different settings in another domain.

Our Learn and Optimize framework (LaO) aims to solve both the intra-domain and extra-domain generalization problems, by adding learning to optimization. We may assume that there might be a relation between some features of the instance and the optimal parameters. If we could learn such a relation represented by a mapping, we could account both for intra-and extra-domain variability of the optimal parameters. The features could describe differences both between instances from the same domain with different complexity, both differences between domains. To do this, we tried to compute features both from the domain-file and the instance-file.

Suppose we have n features and m parameters. For the sake of simplicity and generality, both the fitness value, the features and the parameters are considered as real values. Parameter tuning is the optimization (minimization) of the fitness function $f : \mathbf{R}^n \to \mathbf{R}^m$, which depends on the instance. In our case the expected value off the stochastic algorithm DaE. The the optimal parameter is defined by $p_{opt} = argmin_p\{f(p)\}$. To each instance there is a set F of size n, the features of that instance and domain. There is a possibility that different instances have the same features, but if the feature space is rich enough, this is highly unlikely at least in the representative training set. [2] For the sake of simplicity we consider as if that there existed a (unique) mapping from the feature space to the optimal parameter space.

$$p_F : \mathbf{R}^n \to \mathbf{R}^m, p_F(F) = p_{opt} \quad (1)$$

.

The relation p_F between features and optimal parameters can be learned by any supervised learning method capable of representing, interpolating and extrapolating $\mathbf{R}^n \to \mathbf{R}^m$ mappings.

A simple method could be developed by using any known parameter tuning method for an appropriate training set of instances in a domain, and then use any kind of supervised learning method to learn the relationship of the features and the best parameters. However, we may combine learning and optimizing, and thus we get our LaO algorithm.

It is not a new idea to use some kind of model in optimization. Several so called model-based optimization method exists. In our case, however, there is a difference that we have several instances to optimize, we have only one model, and that model is actually a mapping from the feature-space to the parameter-space. Nevertheless, there is no big issue about how to use the model of p_F in optimization: one can always ask the model for hints of parameters. Naturally, if the model was perfectly fit to the training data, it would return the same hint as trained. Therefore under-fitting is beneficial during the optimization phase. One shall of course also avoid over-fitting in the end.

It seems most reasonable that the stopping criterion of LaO is determined by the stopping criterion of the optimizer algorithm. One can also do a re-training of the learner with the best parameters found.

[2] Moreover, if for both instances the features are the same, we can suppose that there is no big difference in the optimal parameter. Or to be more precise, there is no big difference in fitness when using one or the other optimal parameter, or maybe something "in-between". A learning algorithm would tolerate this and learn something close to the average.

As it is obvious, our LaO algorithm is an open framework: one could use any appropriate learner for the mapping and any kind of optimizer for parameter tuning. LaO can also be generalized to parameter tunning outside of AI planning. In most of the cases, where the parameters of an algorithm are to be tuned, there are instances of application, and in each of these cases there is a possibility to improve the tuning by also learning the relation of some features and the optimal parameters.

Our Implementation of the LaO Framework

Our choice for supervised learning was Artificial Neural Network (ANN), but other algorithms may also be used. We can suppose that the relation p_F is not very complex, which means that a simple ANN may be used. One mapping shall be trained for one domain. One can also try to train one domain independent ANN, but that was not the task in this competition.

The only open decision is which optimizer to use for parameter tuning. Since we have several instances to run, we can only afford a simple optimizer, we choose simply a Covariance Matrix Adaptation Evolution Strategy, specifically, (1+1)-CMA-ES (in short, CMA-ES, see **??**). We started CMA-ES with a known set of good default parameters, see **??**.

There is one element added to a conventional CMA-ES, and this is gene-transfer between instances. We have one CMA-ES running for each instance, because we can not afford to use a larger population. However, the CMA-ES instances of all the instances form a set of individuals. Crossover between individuals is usually not used in ES, however, in our case the the set of individuals does not really resemble to a population, but as a set of species. We have to assume that the optimum for the instances differ, they are in different niche of the parameter tuning word. However, we also may assume that a good chromosome of one instance may cat least help another instance. It may also be used as a hint in the optimization. Therefore random gene-transfer was used in our algorithm.

Using the ANN and gene-transfer as external hints in addition to the CMA-ES corrupts somewhat the later one. The CMA-ES shall be informed about these external hints, if they improve the fitness-function. These external hints are handled as the hint of the CMA-ES algorithm, i.e. we ask for a hint of the CMA-ES and change it to the external hint as if by chance that was the hint of CMA-ES. This way corruption is minimized. The global step size is updated with true or false, depending on the improvement or lack of improvement, covariance matrix is updated only in case of improvement.

There is one additional technical problem with CMA-ES, this is that each parameter is restricted to an interval. This seems reasonable and makes our algorithm more stable. The parameters are actually normalized linearly to the [0,1] interval. For the boundary problem we applied a simple version of the box constraint handling technique described in **??**. Our penalty term was simply $||x^{feas} - x||$, where x^{feas} is the closest value in the box. Moreover, only x^{feas} was recorded as a feasible solution, to pass for example to the ANN. Note that gene-transfer and the ANN itself can not give hints out of the box. In order to not to compromise too much CMA-ES several iterations of this were carried out for one hint of the ANN and one gene-transfer.

Our realization of our LaO algorithm uses the Shark library (**??**) for CMA-ES and the FANN library for ANN (**??**). To evaluate each parameter-setting with each instance, we use a computer-cluster of LRI with 60 computers, most of them have 4 cores, but some have 8. The cluster is used by many researchers, so our algorithm contains a scheduler to use the free capacity on this cluster.

Since the parallel architecture used in our algorithm is somewhat heterogeneous, we can not rely on a fixed running time. Therefore, for each evaluation the number of evaluations is fixed for DaE. Moreover, since DaE is not deterministic, we carried out 11 runs and took the median cost as the result for that instance and that parameter-settings.

Acknowledgements

Restrictions on Further Use

References

Bibai, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2008. DAE : Planning as Artificial Evolution (Deterministic part). At International Planning Competition (IPC) `http://ipc.icaps-conference.org/`.

Bibai, J.; Savéant, P.; Schoenauer, M.; and Vidal, V. 2010. On the generality of parameter tuning in evolutionary planning. In et al., J. B., ed., *Genetic and Evolutionary Computation Conference (GECCO)*, 241–248. ACM Press.

Bibai, J.; Savéant, P.; and Schoenauer, M. 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.

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

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

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

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

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

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

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

Jacques Bibai; Pierre Savéant; Marc Schoenauer; and Vincent Vidal. 2010a. An evolutionary metaheuristic based on state decomposition for domain-independent satisficing planning. In *ICAPS 2010*, 18–25. AAAI press.

Jacques Bibai; Pierre Savéant; Marc Schoenauer; and Vincent Vidal. 2010b. On the benefit of sub-optimality within the divide-and-evolve scheme. In Cowling, P., and Merz, P., eds., *EvoCOP 2010*, number 6022 in Lecture Notes in Computer Science, 23–34. Springer-Verlag.

Lobo, F.; Lima, C.; and Michalewicz, Z., eds. 2007. *Parameter Setting in Evolutionary Algorithms*. Berlin: Springer.

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

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.

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

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*, 150–159. Whistler, BC, Canada: AAAI Press.

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

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

Yu, T.; Davis, L.; Baydar, C.; and Roy, R., eds. 2008. *Evolutionary Computation in Practice*. Studies in Computational Intelligence 88, Springer Verlag.