Optimize-and-Learn: a Parameter Tuning Framework applied for the Divide-and-Evolve Method used to AI PLanning

Mátyás Brendel

Projet TAO
INRIA Saclay & LRI
Université Paris Sud
Orsay, France
matthias.brendel@lri.fr

Marc Schoenauer

Projet TAO
INRIA Saclay & LRI
Université Paris Sud
Orsay, France
marc.schoenauer@inria.fr

Abstract

In this paper we describe the system used in the Planning and Learning Part of the 7th International Planning Competition. Learn-and-Optimize (LaO) is a generic surrogate based method for parameter tuning combining learning and optimization. In this paper LaO is used to tune Divide-and-Evolve (DaE), an Evolutionary Algorithm for AI Planning. The LaO framework makes it possible to learn the relation between some features describing a given instance and the optimal parameters for this instance, thus it enables to extrapolate this relation to unknown instances in the same domain. Moreover, the learned model is used as a surrogate-model to accelerate the search for the optimal parameters. It hence becomes possible to solve intra-domain and extra-domain generalization in a single framework. The proposed implementation of LaO uses an Artificial Neural Network for learning the mapping between features and optimal parameters, and the Covariance Matrix Adaptation Evolution Strategy for optimization.

Introduction

Parameter tuning is basically a general optimization problem applied off-line to find the best parameters for complex algorithms, for example for Evolutionary Algorithms (EAs). Whereas the efficiency of EAs has been demonstrated on several application domains (Yu et al. 2008; Lobo, Lima, and Michalewicz 2007), they usually need computationally expensive parameter tuning. Consequently, one is tempted to use either the default parameters of the framework he is using, or parameter values given in the literature for problems that are similar to his one.

There may be as many parameter tuning algorithms as optimization techniques (Eiben et al. 2007), (Montero, Riff, and Neveu 2010). However, several specialized methods have been proposed, and the most prominent today are Racing (Birattari et al. 2002), REVAC (Nannen, Smit, and Eiben 2008), SPO (Bartz-Beielstein, Lasarczyk, and Preuss 2005), and ParamILS (Hutter et al. 2009). All these approaches face the same crucial generalization issue: can a parameter set that has been optimized for a given problem be successfully used to another one? The answer of course depends on the similarity of both problems.

Unfortunately, until now, nobody has yet proposed a set of features for AI Planning problems in general, that would be sufficient to describe the characteristics of a problem, like it was done in the SAT domain (Hutter et al. 2006). This paper makes a step toward a framework for parameter tuning applied generally for AI Planning and proposes a preliminary set of features. The Learn-and-Optimize (LaO) framework consists of the combination of optimizing (i.e., parameter tuning) and learning, i.e., finding the mapping between features and best parameters. Furthermore, the reuslts of learning will already be useful during further the optimization phases, using the learned model as in standard surrogate-model based techniques.

In this paper, the target optimization technique is Evolutionary Algorithms (EA), more precisely the evolutionary AI planner called Divide-and-Evolve (DaE). However, DaE will be here considered as a black-box algorithm, without any modification for the purpose of this work than its original version described in (Jacques Bibai et al. 2010b).

The paper is organized as follows: Section describes the evolutionary Divide-and-Evolve algorithm. Section introduces the original, top level parameter tuning method, Learn-and-Optimize. Finally, section describes the implementation of the framework used in the Planning and Learning Part of IPC2011.

Divide-and-Evolve

Divide-and-Evolve (DaE) – a novel hybridization of Evolutionary Algorithms (EAs) with AI Planning – has been first proposed in (Schoenauer, Savéant, and Vidal 2006). For a complete formal description, see (Jacques Bibai et al. 2010a).

A planning problem defined on domain D with initial state I and goal G will be denoted in the following as $\mathcal{P}_D(I, G)$. The basic idea of DaE in order to solve a planning task $\mathcal{P}_D(I, G)$ 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 $(S_i)_{i\in[1,n]}$ 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 the external 'embedded' planner to solve the sequence of problems $\mathcal{P}_D(S_k, S_{k+1})$, $\{k = 0, \ldots, n\}$. The concatenation of the corresponding plans (possibly with some compression step) is a solution of the initial problem. A sub-optimal, but fast planner is used as an embedded planner: YAHSP (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.

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 be to 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.

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, as far as the initial grounding of all atoms can tell (exactly determining if two atoms are mutex amounts to solving a complete planning problem). Furthermore, all operators that manipulate the representation (see below) maintain the chronology between atoms and the approximate local consistency of a state, i.e. avoid pairwise mutexes.

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. The complete list of DaE parameters that required some tuning is given in Table 1.

The General LaO Framework

If parameter tuning is applied to AI Planning, tuning one instance has of course a sense if only that instance is to be solved. Parameters tuned for one instance however, may not be optimal for other instances, as (Bibai et al. 2010) demonstrates it. This very same paper also demonstrates that global tuning for several domains is even more inferior.

Our Learn-and-Optimize framework (LaO) aims to answer this question and to solve intra-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 representable 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, both differences between domains. To do this, we try to extract 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 (in our case minimization) of the fitness function $f : \mathbf{R}^m \to \mathbf{R}$, which function is different for each instance. In our case f is the expected value off the stochastic algorithm DaE executed with parameter $p \in \mathbf{R}^m$. The optimal parameter is defined by $p_{opt} = argmin_p\{f(p)\}$. For each instance there is a set $F \in \mathbf{R}^n$, the features of that instance and the corresponding domain.

We suppose that there exists an "almost unambiguous" 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. The learning method shall also be capable to resolve the unambiguity of the mapping.

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 an appropriate supervised learning method to learn the relationship of the features and the best parameters. However, we can do more: learning and optimizing may be combined, and thus we get our LaO algorithm.

It is not a new idea to use some kind of model in optimization. Several so called surrogate-model based optimization methods exist. 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 question about how to use such a 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 be of no use, since it would return the same hint as trained. Therefore under-fitting is beneficial during the optimization phase to get new hints. One shall of course avoid over-fitting also in the end, but in the end we can not allow under-fitting.

It seems most reasonable that the stopping criterion of LaO is determined by the stopping criterion of the optimizer algorithm. After exiting one can also do a retraining of the learner with the best parameters found.

An Implementation of LaO

Our choice for supervised learning method was a multilayer Feed-Forward 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 a single domain-independent ANN, but that will be left to the future.

Since we have several instances to run, we can only afford a simple optimizer, therefore simply a Covariance Matrix Adaptation Evolution Strategy was chosen, specifically, a (1+1)-CMA-ES (in short, CMA-ES, see (Hansen and Ostermeier 2001)). We started CMA-ES with a known set of good default parameters, taken from (Bibai et al. 2010).

There is one element added to a conventional CMA-ES, and this is gene-transfer between instances. We have one (1+1)-CMA-ES running for each instance, because we can not afford to use a larger population for a single instance. However, the (1+1)-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 a good chromosome of one instance may at least help another instance. Thus it may be used as a hint in the optimization. Therefore random gene-transfer was used in our algorithm. When the Gene-transferer is requested for a hint for one instance, it returns with uniform random distribution the so-far best parameter of a different instance. Naturally, the default parameters are not tried twice.

Our realization of our LaO algorithm uses the Shark library ((Igel, Glasmachers, and Heidrich-Meisner 2008)) for CMA-ES and the FANN library for ANN ((Nissen 2003)). To evaluate each parameter-setting with each instance, we use a computer-cluster with approximately 60 computers, most of them have 4 cores, but some have 8. The cluster is used by many researchers, therefore 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, which depends on the hardware. Therefore, for each evaluation the number of YAHSP evaluations is fixed for DaE. Moreover, since DaE is not deterministic, we carried out 11 runs and took the median fitness-value as the result for that instance and that parameter-settings.

Name	Min	Max	Default
Probability of crossover	0.0	1	0.8
Probability of mutation	0.0	1	0.2
Rate of mut. add station	0	10	1
Rate of mut. delete stat.	0	10	3
Rate of mut. add atom	0	10	1
Rate of mut. delete atom	0	10	1
Mean average for mut.	0.0	1	0.8
Time interval radius	0	10	2
Max. number of stations	5	50	20
Max. number of nodes	100	100 000	10 000
Population size	10	300	100
Number of offsprings	100	2 000	700

Table 1: DaE parameters that are controlled by LaO

5 iterations of CMA-ES were carried out, followed by one ANN and one Genetransferer, and this cycle was iterated in the algorithm. The ANN had 3 fully connected layers, and the hidden layer had the same number of neurons as the input. Learning was done by the conventional back-propagation algorithm, which is the default in FANN. In one iterations of LaO the ANN was only trained once for 50 iterations (called epochs in FANN) without reseting the weights, so that we avoid over-training. The aim of not reseting the weights was that the ANN makes a graded transition from the previous best known parameter-set to the new best known parameter-set, which could help optimization by trying some intermediate values. This is why retraining of the ANN is needed in the end.

Termination criterion in the competition was simply the available time, the algorithm was running for several

weeks on our cluster, which is used also for other research, i.e. only a small number of 4 or 8-core processors were available for each domain in average. After stopping LaO, retraining was made with 300 ANN epochs with the best data, because the ANN's saved directly from LaO may be under-trained. The MSE error of the ANN did not decrease using more epochs, which indicates that 300 iterations are enough at least for this amount of data and for this size of the ANN. Tests with 1000 iterations did not produce better results and neither training the ANN uniquely with the first found best parameters.

The controlled parameters of DaE are described in table 1. For a detailed description of these parameters, see (Bibai et al. 2010). The feature-set consists of 12 features. First there are 5 important features: the number of fluents, goals, predicates, objects and types. These were extracted from the domain file or the instance file using the PDDL parser. One further feature we think could even be more important is called mutex-density, which is the number of mutexes divided by the number of all fluent-pairs. We also kept 6 less important features: number of lines, words and byte-count - obtained by the linux command "wc" - of the instance and the domain file. These features were kept only for historical reasons: they were used in the beginning as some "dummy" features.

Acknowledgements

This work is funded through French ANR project DESCARWIN ANR-09-COSI-002.

References

Bartz-Beielstein, T.; Lasarczyk, C.; and Preuss, M. 2005. Sequential parameter optimization. In McKay, B., ed., *Proc. CEC'05*, 773–780. IEEE Press.

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.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A Racing Algorithm for Configuring Metaheuristics. In *GECCO '02*, 11–18. Morgan Kaufmann.

Eiben, A. E.; Michalewicz, Z.; Schoenauer, M.; and Smith, J. E. 2007. Parameter control in evolutionary algorithms. In Lipcoll et al. (2007). chapter 2, 19–46.

Hansen, N., and Ostermeier, A. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9(2):159–195.

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.

Igel, C.; Glasmachers, T.; and Heidrich-Meisner, V. 2008. Shark. *Journal of Machine Learning Research* 9:993–996.

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.

Montero, E.; Riff, M.-C.; and Neveu, B. 2010. An evaluation of off-line calibration techniques for evolutionary algorithms. In *Proc. ACM-GECCO*, 299–300. ACM.

Nannen, V.; Smit, S. K.; and Eiben, A. E. 2008. Costs and benefits of tuning parameters of evolutionary algorithms. In *Proceedings of the 20th Conference on Parallel Problem Solving from Nature*.

Nissen, N. 2003. Implementation of a Fast Artificial Neural Network Library (FANN). Technical report, Department of Computer Science University of Copenhagen (DIKU).

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.

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.

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