Instance-based parameter tuning for evolutionary AI planning

Brendel Matthias, Marc Schoenauer

Instance-Based Parameter Tuning for Evolutionary 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

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 knowledge to unknown instances in the same domain. Moreover, the learned relation 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. Results demonstrate that LaO is capable of improving the quality of the DaE results even with only a few iterations. The main limitation of the DaE case-study is the limited amount of meaningful features that are available to describe the instances. However, the learned model reaches almost the same performance on the test instances, which means that it is capable of generalization.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence Learning Parameter learning

General Terms

Theory

Keywords

parameter tuning, AI Planning, evolutionary algorithms

1. INTRODUCTION

GECCO'11, July 12–16, 2011, Dublin, Ireland.

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 [29, 18], 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.

Being a general optimization problem, there are as many parameter tuning algorithms as optimization techniques [7, 19]. However, several specialized methods have been proposed, and the most prominent ones today are Racing [5], REVAC [21], SPO [2], and ParamILS [14]. All these approaches face the same crucial generalization issue: can a parameter set that has been optimized for a given problem be successfully used for another one? The answer of course depends on the similarity of both problems. However, even in an optimization domain as precisely defined as AI Planning, there are very few results describing meaningful similarity measures between problem instances. Moreover, until now, sufficiently precise and accurate features have not been specified that would allow the user to accurately describe the problem, so that the optimal parameter-set could be learned from this feature-set, and carried on to other problems with similar description. To the best of our knowledge, no design of a general learning framework has been proposed, and no general experiments have been carried out yet with some representative domains of AI planning.

In the SAT domain, however, one work must be given as an example of what can be done along those lines. In [13], 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 has allowed the authors to learn, using function regression, a meaningful mapping between the features and the running-time of a given SAT solver with given parameters. Optimizing this model makes it possible to choose the optimal parameters for a given (unknown) instance. The present paper aims at generalizing this work made in AI planning, with one major difference: the target will be here to optimize the fitness value for a given runtime, and not the runtime to solution – as the optimal solution is generally not known for AI planning 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 was done in the SAT domain [13]. This paper makes a

step toward a framework for parameter tuning applied generally to 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 results of learning will already be useful to further the optimization phases, using the learned model as in standard surrogate-model based techniques (see e.g., [1] for a Gaussian-process-based approach). LaO can of course be applied to any target optimization methodology that requires parameter tuning. In this paper, the target optimization technique is an Evolutionary Algorithm (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 compared to its original version described in [17].

The paper is organized as follows: AI Planning Problems are briefly introduced in section 2. Section 3 describes the and the classical YAHSP solver and the evolutionary Divide-and-Evolve algorithm. Section 4 introduces the original, top level parameter tuning method, Learn-and-Optimize. The case study presented in Section 5 applies LaO to DaE, following the rules of the International Planning Competition 2011 – Learning Track. Finally, conclusions are drawn and further directions of research are proposed in Section 6.

2. AI PLANNING

An Artificial Intelligence (AI) planning problem 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 plan to a planning problem 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 the number of actions; it may also be the total cost of the pan for actions with cost; and it is the total duration of the plan, aka *makespan*, for temporal problems with so called durative actions.

Domain-independent planners rely on the Planning Domain Definition Language PDDL2.1 [8]. 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, written in PDDL, are still the main benchmarks in AI Planning. The description of a planning problem consists of two separate parts usually placed in two different files: the generic domain on the one hand and a specific instance scenario on the other hand. The domain file specifies object types and predicates, which define possible states, and actions, which define possible state changes. The instance scenario declares the actual 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 plan to a planning problem is a consistent schedule of grounded actions whose execution in the initial state leads to a state that contains the goal state, i.e., where all atoms of the problem goal are true. A planning problem defined on domain D with initial state I and goal G will be denoted in the following as $\mathcal{P}_D(I, G)$.

3. DIVIDE-AND-EVOLVE

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 [25, 20, 27, 28, 6]. However, as it is often the case in Evolutionary Combinatorial optimization, those direct encoding approaches have limited performance in comparison to the traditional AI planning approaches. Furthermore, hybridization with classical methods has been the way to success in many combinatorial domains, as witnessed by the fruitful emerging domain of memetic algorithms [11]. 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 [23, 24]. For a complete formal description, see [16].

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 (makespan or total cost) 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. Any existing planner can be used as embedded planner, but since guaranty of optimality at all calls is not mandatory in order for DaE to obtain good quality results [16], a sub-optimal, but fast planner is used: YAHSP [26] 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 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 [3] 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... [12]). 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).
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 list of DaE parameters that will be tuned in this paper is given in Table 2.

4. LEARN-AND-OPTIMIZE FOR PARAMETER TUNING

4.1 The General LaO Framework

As already mentioned, parameter tuning is actually a general global optimization problem, thus facing the routine issue of local optimality. But a further problem arises in parameter tuning, and this is the generality of the tuned parameters. Tuning only 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 [4] demonstrates. Furthermore, this paper also demonstrates that parameter tuning for several domains simultaneously is even more difficult, if at all possible.
Even generalizing parameters learned on one instance to another instance of the same domain (intra-domain generalization) might be problematic, as there are instances with very different complexity in the same domain. For example in [4] per-domain tuning was performed with the most difficult, largest instance, considered as a representative of the whole domain. However, it is clear from the results that these parameters were often suboptimal for the other instances. And similar issues might arise even when using several instances as representatives of the domain. Since the optimum values of the parameters might change from instance to instance, only a "dull" average-like parameter-setting may be computed. Moreover, the computational cost of parameter tuning increases linearly with the number of training instances.
The issue is of course even more critical when aiming at inter-domain generalization, i.e., learning the parameters on one or several instances, and using the learned parameters on instances of different domain than that of the training instances. Indeed, differences between the domains may cause a problem, and even instances of apparent similar complexity (e.g. same number of objects) may require different settings from domain to domain. The poor results with global tuning in [4] indicate that these are issues to be considered.
One workaround for this generalization issue is to relax the constraint of finding a single universally optimal parameter-set - that certainly does not exist - and to focus on learning a complex relation between instances and optimal parameters. Moreover, to be able to generalize to new instances, the instances have to be represented by features, and the relation between features and parameters has to be learned.
The proposed Learn-and-Optimize framework (LaO) aims at learning such a relation, thus, in the ideal case, solving both the intra-domain and extra-domain generalization problems. The underlying hypothesis is that there exists a relation between some features describing an instance and the optimal parameters for solving this instance which can be learned, and the goal of this work is to propose a general methodology to do so. If well designed, the features should describe differences both between instances from the same domain, and differences between instances of different domains – and hence differences between domains, too. The case study analyzed here deals with AI planning, and some features extracted from both the domain-file and the instance-file will be proposed later.
Suppose for now that we have n features and m parameters, and we are doing per-instance parameter tuning on instance $\mathcal{I}$. For the sake of simplicity and generality, both the fitness, the features and the parameters are considered as real values. Parameter tuning is the optimization (e.g., minimization) of the fitness function $f_{\mathcal{I}} : \mathbf{R}^m \to \mathbf{R}$, the expected value of the stochastic algorithm DaE executed with parameter $p \in \mathbf{R}^m$. The optimal parameter set is defined by $p_{opt} = argmin_p\{f_{\mathcal{I}}(p)\}$.
For each instance $\mathcal{I}$, consider the tuple $F(\mathcal{I}) \in \mathbf{R}^n$ of the features describing this instance. Two relations have to be taken into account: each planning instance has features, and it has an optimal parameter-set. In order to be able to generalize, we have to get rid of the instance, and collapse both relations into one single relation between feature-space and parameter-space. This is only possible if different instances have different features, or, more generally, if instances having the same features also share the same optimal parameter set. Relaxing this tight constraint, we could simply assume that the features are such that if two instances have similar features, their optimal parameter sets is such that using either sets does not make a big difference regarding the optimization problem (i.e., for AI planning, leads to similar makespans). However, for the sake of simplicity let us omit the dependency to I, and assume that there exists an unambiguous mapping from the feature space to the optimal parameter space.

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

.
Nevertheless, we will indicate, if some problems with the results may be caused by an unambiguity. The relation p 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, provided sufficient data are available.

A simple method could use any standard parameter tuning method for an appropriate training set of instances in a given domain, and then to use an appropriate supervised learning method in order to learn the relationship between the features and the best parameters. Moreover, learning and optimizing may be combined, and this is the second main idea behind LaO.

The idea of using some surrogate model in optimization is not new. In our case, however, there are several instances to optimize, and only one model is available, that maps the feature-space into the parameter-space. Nevertheless, there is no question about how to use such a model of p in optimization: one can always ask the model for hints about a given parameter-set. Of course, if the model were perfectly fit to the training data, it would be useless, since it would return the same hint as trained. Therefore under-fitting when learning the mapping from feature-space to parameter-space is beneficial during the optimization phase in order to get new hints.

It seems reasonable that the stopping criterion of LaO is determined by the stopping criterion of the optimizer algorithm. After exiting one can also do a re-training of the learner with the best parameters found. One shall of course also avoid in this case the other regular threat on learning algorithms, that is over-fitting.

The proposed 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 of course be generalized to parameter tuning outside of AI planning. In most 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 between some features and the optimal parameters.

4.2 An Implementation of LaO

A simple multilayer Feed-Forward Artificial Neural Network (ANN) trained with standard backpropagation was chosen here for the learning of the features-to-parameters mapping, though any other supervised-learning algorithm could have been used. The implicit hypothesis is that the relation p is not very complex, which means that a simple ANN may be used. In this work, one mapping is trained for each domain. Training a single domain-independent ANN is left for future work.

The other decision for the LaO implementation is the choice of the optimizer used for parameter tuning. As described in the introduction, one could use for this sake REVAC, SPO, RACING or ParamILS. Because however, the parameter optimization will be done successively for several instances, the simple yet robust (1+1)-Covariance Matrix Adaptation Evolution Strategy [10], in short (1+1)-CMA-ES, was chosen, and used with its robust own default parameters, as advocated in [4]. CMA-ES is a maximum-likelihood based method estimating the so called natural gradient and improving efficiency this way over the conventional gradient-based methods.

One original component, though, was added to some direct approach to parameter tuning: gene-transfer between instances. There will be one (1+1)-CMA-ES running for each instance, because using larger population sizes for a single instance would be far too costly. However, the (1+1)-CMA-ES algorithms running on all training instances form a population of individuals. The idea of gene-transfer is to use something similar to crossover between the individuals of this population. Gene-transfer was chosen instead of crossover, because there is a high chance that a parameter-set good for one instance is good for the other as it is. Thus it may be used as a hint in the optimization of that other instance. Therefore random gene-transfer was used in the present implementation of LaO, by calling the so-called *Genetransferer*. When the Genetransferer is requested for a hint for one instance, it returns the so-far best parameter of a different instance chosen with uniform random distribution (preventing, of course, that the default parameters are tried twice). Another benefit of the Genetransferer is that it may smoothen out the ambiguities between instances, by increasing the probability for instances with the same features to test the same parameters, and thus the possibility to find out that the same parameters are appropriate for the same features. Figure 1 shows the LAO framework with the described implementations, and Algorithm 1 shows the pseudo-code. In this pseudo-code for easier comprehensibility, we did not include the test of the default parameters, the case of multiple best parameters and the modification introduced below for increasing the integrity of CMA-ES are also not shown.

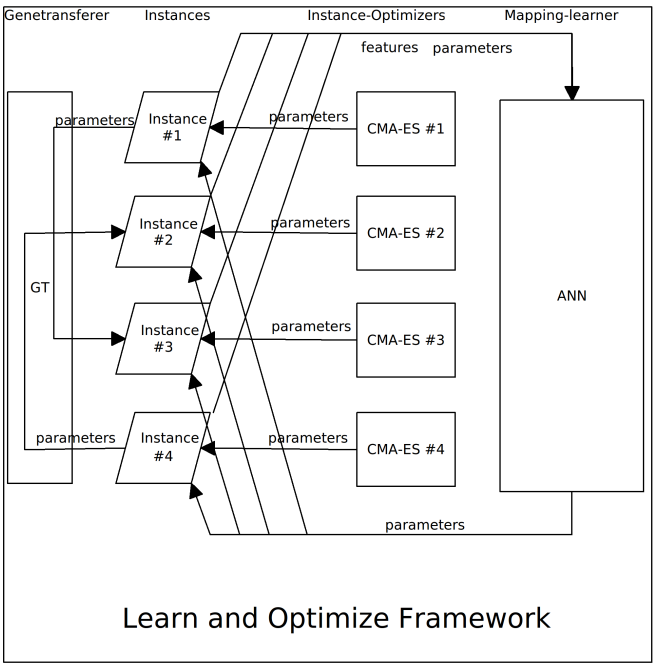

Figure 1: Flowchart of the Lao framework, displaying only 4 instances.

Care must be taken when using the ANN and the Genetransferer as external hints within the standard CMA-ES process, to avoid corrupting it too much. Indeed, CMA-ES should be informed about these external hints, if they improve the fitness-function. The proposed solution is to handle those parameter-sets as if they were by chance the hint of the CMA-ES algorithm, i.e. to replace a standard request from CMA-ES by the value of the external hint, thus minimizing possible corruption. The global step size is updated with true or false, depending on the improvement or lack of improvement, and as in the usual CMA-ES algo-

Domain Name	# of iterations	# training instances	# test instances	ANN error	quality-ratio in LaO	quality-ratio ANN on train	quality-ratio ANN on test
Freecell	16	108	230	0.1	1.09	1.05	1.04
Grid	10	55	124	0.09	1.09	1.05	1.03
Mprime	8	64	152	0.08	1.11	1.05	1.04

Table 1: Results by domains (only the actually usable training instances are shown). ANN-error is given as MSE, as returned by FANN. The quality-improvement ratio in Lao is that of the best parameter-set found by LaO.

Algorithm 1 learn-and-optimize()

Require: #cma, #epochs, instances
while exitCriterionFalse() **do**
  for $c = 1 \rightarrow \#cma$ **do**
    for all $I \in instances$ **do**
      $p \leftarrow I.callCMA()$ //each instance has its own CMA
      $f \leftarrow I.evaluate(p)$ //also keeping track of best p
      $I.updateCMA(f)$
    $c \leftarrow c + 1$
  for all $I \in instances$ **do**
    $I^* \leftarrow callGenetransferer(I)$ //a different instance
    $p \leftarrow I^*.getBestParameter()$
    $f \leftarrow I.evaluate(p)$
  for all $I \in instances$ **do**
    $p \leftarrow I.getBestParameter()$
    $F \leftarrow I.getFeatures()$
    $addANN(F, p)$
  $trainANN(\#epochs)$
  for all $I \in instances$ **do**
    $F \leftarrow I.getFeatures()$
    $p \leftarrow callANN(F)$
    $f \leftarrow I.evaluate(p)$
return

rithm, the covariance matrix is updated only in the case of improvement.

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

Table 2: DaE parameters that are controlled by LaO

One additional technical difficulty arose with CMA-ES: each parameter is here restricted to an interval. This seems reasonable and makes the global algorithm more stable. Hence the parameters of the optimizer are actually normalized linearly onto the [0,1] interval. It is hence possible to apply a simple version of the box constraint handling technique described in [9], with a penalty term simply defined by $||p^{feas} - p||$, where p^{feas} is the closest value in the box. Moreover, only p^{feas} was recorded as a feasible solution, and later passed to the ANN. Note that the GeneTransferer and the ANN itself cannot return hints outside 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 Genetransferer.

The implementation of LaO algorithm uses the Shark library [15] for CMA-ES and the FANN library for ANN [22]. To evaluate each parameter-setting with each instance, a cluster was used, that has approximately 60 nodes, most of them with 4 cores, some with 8. However, this cluster is used by many researchers, therefore our algorithm was automatically scheduled to only use the spare CPU cycles on this cluster. Because of the heterogeneity of the hardware architecture used here, it is not possible to rely on accurate predicted running times. Therefore, for each evaluation, the number of YAHSP evaluations in DaE is fixed. Note that the number of YAHSP evaluations is approximately proportional to the running time, so that the execution time for a particular computer is also determined independently of the parameter-settings. For example, even if the size of the population is increased, because of the fixed number of evaluations that is allowed, the number of generations will be limited accordingly in order to approximatively allow the same running time for each run of DaE with each parameter-setting.

Moreover, since DaE is not deterministic, 11 independent runs were carried out for each DaE experiment with a given parameter-set, and the fitness of this parameter set was taken to be the median.

5. RESULTS

In the Planning and Learning Part of IPC2011 (IPC), 5 sample domains were pre-published, with a corresponding problem-generator for each domain: Ferry, Freecell, Grid, Mprime, and Sokoban. Ferry and Sokoban were excluded from this study since there were not enough number of instances to learn any mapping. For each of the remaining 3 domains, 100 instances were generated, by the published generators and distributions. 100 instances seemed to be appropriate for a running time of approximately 2-3 weeks. The competition track description fixes running time as 15 minutes. For each instance, 11 independent trials were run on a dedicated server to measure the median of number of YAHSP-evaluations with the default parameters. Subsequently, this number of evaluations was used for DaE as a termination criterion on any computer in LaO. However, many instances were never solved within 15 minutes, and those instances were dropped from the rest of experiment. The remaining instances were used for training.

Table 1 presents the train and test for each domain: from the 5 domain, only 3 had enough solvable instances to be used for the learning part. The Mean Square Error (MSE) of the trained ANN is shown for each domain. But because the fitness takes only few values, there can be multiple optimal parameter sets for the same instance, resulting in an unavoidable MSE. One iteration of LaO amounts to 5 iter-

Name	Default	CMA-ES	Transferer	ANN
Freecell	0 – 9	64 – 66	18 – 8	18 – 17
Grid	2 – 24	66 – 60	16 – 11	17 – 5
Mprime	2 – 45	59 – 36	2 – 11	18 – 8

Table 3: For each method (default, CMA-ES, Genetransferer or ANN), the percentage of instances on which this method gave the best parameter set. Each cell shows 2 figures: the first one considers all occurrences of a method, no matter if another method also lead an equivalent parameter set, as good as the first one. The second figures only considers the first method (from left to right) that discovered the best parameter-set.

ations of CMA-ES, followed by one ANN training and one Genetransferer. Due to the time constraints, only few iterations of LaO were run. For example in domain Grid only 10 and CMA-ES was called 50 times in total.
The ANN had 3 fully connected layers, the layers had all 12 neurons, corresponding to the number or parameters and features, respectively. Standard back-propagation algorithm was used for learning (the default in FANN). In one iteration of LaO, the ANN was only trained for 50 iterations (aka epochs) without reseting the weights, in order to i- avoid over-training, and ii- making a gradual transition from the previous best parameter-set to the new best one, and eventually try some intermediate values. Hence, in domain Grid, over the 10 iterations of LaO, 500 iterations (epochs) of the ANN were carried out in total. However, note that the best parameters were trained with much fewer iterations, depending on the time of their discovery. In the worst case, if the best parameter was found in the last iteration of LaO, it was trained for only 50 epochs (and not used anymore). This explains why retraining is needed in the end.
A parameter-set in LaO may come from different sources, namely it can be the default parameter-set, or coming from CMA-ES, the Genetransferer, or as a result of applying the trained ANN to the instance features. Table 3 shows how each source contributes to the best overall parameter-settings. For each possible source, the first number is the ratio the source contributed to the best result if tie-breaks are taken into account, the second number shows the same, if only the first best parameter-set is taken into account. Note that the order of the calling of the "sources" in LaO is the same as it is in the table: for example if CMA-ES found a different parameter-settings with the same fitness than the default, this case is not included in the first ratio, but is in the second. Analyzing both numbers leads to the following conclusions: for domain Mprime, the default parameter-settings was the optimal for 45% of the instances. However, only in 2% of the instances there was no other parameter-setting found with the same quality. The reason for this is that makespan values for Mprime are mostly single digit numbers. Consequently, there is no possibility for a small improvement, an improvement is more rare (55%) but those improvement are naturally high in ratio. In the domain Freecell, the share of ANN is quite high (18%), moreover we can see that in most cases, the other sources did not find a parameter-set with the same performance (17%). While Genetransferer in Freecell take equal share (18%) of all the best parameters, but only a part of them (8%) were unique. Note that CMA-ES was returning the first hint in each iteration and had 5 times more possibilities than the ANN. Taking this into account, it is clear that both the ANN and Genetransferer made an important contribution to optimization.
LaO has been running for several weeks on a cluster. But this cluster was not dedicated to our experiments, i.e. only a small number of 4 or 8-core processors were available for each domain on 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 2. For a detailed description of these parameters, see [4]. The feature-set consists of 12 features. The first 5 features are computed from the domain file, after the initial grounding of YAHSP: number of fluents, goals, predicates, objects and types. 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. Since mutexes are kind of obstacles in planning, higher density indicates more difficulty in finding the solution. 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.
Since testing was also carried out on the cluster, the termination criterion for testing was also the number of evaluations for each instance. For evaluation the quality-improvement the quality-ratio metric defined in IPC competitions was used. The baseline qualities come from the default parameter-setting. The ratio of the fitness value for the default parameter and the tuned parameter was computed and average was taken over the instances in the train or test-set.

$$Q = \frac{Fitness_{baseline}}{Fitness_{tuned}} \quad (2)$$

Note that there was no unsolved instance in the training set, because they were dropped from the experiment if they were not solved with the default parameters.
Table 1 presents several quality-improvement ratios. Label "in LaO" means that the best found parameter is compared to the default. By definition, this ratio can never be less than 1, because the default values are the starting point of the optimizations. This improvement indicated by high quality-ratio is already useful if the very same instances used in training have to be optimized. Quality-improvement ratios for the retrained ANN on both the training-set and the test-set are also presented. In these later cases, numbers less then 1 are possible (the parameters resulting from the retrained ANN can have worse results than the ones given by the default parameters), but were rare. As it can be seen in table 1, some quality-gain in training was consistently achieved, but the transfer of this improvement to the ANN-model was only partial. The phenomenon can appear because i- of the unambiguity of the mapping, or because ii- the ANN is not complex enough for the mapping, or, and most probably, because the feature-set is not representative enough.

On the other hand, the ANN model generalizes excellently to the independent test-set. Quality-improvement ratios dropped only by 0.01, i.e. the knowledge incorporated in the ANN was transferable to the test cases and usable almost to the same extent than for the train set.
The results are quite similar for all domains. Even the size of the training set seems not to be so crucial. For example for Freecell all the instances (108 out of 108 generated) could be used, because they were not so hard. On the other hand, only few Grid instances (55 out of 107 generated) could be used. However, both performed well. The explanation for this may be that both the 32 and 108 instances covered well the whole range of solvable instances.

6. CONCLUSIONS AND FUTURE WORK

The LaO method presented in this paper is a surrogate-model based combined learner and optimizer for parameter tuning. LaO was demonstrated to be capable of improving the quality of the DaE algorithm consistently, even though it was run only for a few iterations. Ongoing work is concerned with running LaO for an appropriate number of iterations. A clearly visible result is also that some of this quality-improvement can be incorporated into an ANN-model, which is also able to generalize excellently to an independent test-set.
Of course, LaO also has its own parameters, which should be tuned, too. Parameter tuning is in this respect similar to the question of the final parameters of an ultimate theory of the Universe: one can always try to reduce one theory to another one that possibly has less parameters. But such infinite regress can not be stopped: there will always be some ultimate parameters to be tuned. Similarly, the parameters of any algorithm can be tuned by another algorithm and that may improve the results. However, there is no ultimate algorithm. The bad news here is that the computing capacity needed is combinatorially exploding. The good news is that in each step in the hierarchy an improvement can be made. Nevertheless, the possible improvements become smaller and smaller.
The most important experiment to carry out in the future is simply to test the algorithm with more iterations and on more domains – and this will take several months of CPU even using a large cluster. Since LaO is only a framework, as indicated other kind of learning methods, and other kind of optimization techniques may be incorporated. If an ANN is used, the optimal structure has to be determined, or a more sophisticated solution is to be applied, one of the so-called Growing Neural Network architectures.
Also the benefit of gene-transfer and/or crossover should be investigated further. Gene-transfer shall be improved so that chromosomes are transfered deterministically, measuring the similarity of instances by the similarity of their features.
Finally, the inter-domain generalization capability of LaO must be tested: It might be possible to learn a mapping for many domains, since the features may grasp the specificity of a domain, and several domains might share some specificities. However, the present results indicate that the current feature set is too small and should be extended for better results. Feature-selection would then become important only if the number of features is large compared to the number of examples. Unfortunately, this is not the case yet.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] R. Bardenet and B. Kégl. Surrogating the surrogate: accelerating gaussian-process-based global optimization with a mixture cross-entropy algorithm. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, 2010.

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

[3] J. Bibai, P. Savéant, and M. Schoenauer. Divide-And-Evolve Facing State-of-the-Art Temporal Planners during the 6^{th} International Planning Competition. In C. Cotta and P. Cowling, editors, *EvoCOP'09)*, number 5482 in LNCS, pages 133–144. Springer-Verlag, 2009.

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

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

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

[7] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In Lipcoll et al. [18], chapter 2, pages 19–46.

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

[9] N. Hansen, S. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009.

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

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

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

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

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

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

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

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

[18] F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, Berlin, 2007.

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

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

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

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

[23] M. Schoenauer, P. Savéant, and V. Vidal. Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In J. Gottlieb and G. Raidl, editors, *Proc. EvoCOP'06*. Springer Verlag, 2006.

[24] M. Schoenauer, P. Savéant, and V. Vidal. Divide-and-Evolve: a Sequential Hybridization Strategy using Evolutionary Algorithms. In Z. Michalewicz and P. Siarry, editors, *Advances in Metaheuristics for Hard Optimization*, pages 179–198. Springer, 2007.

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

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

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

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

[29] T. Yu, L. Davis, C. Baydar, and R. Roy, editors. *Evolutionary Computation in Practice*. Studies in Computational Intelligence 88, Springer Verlag, 2008.