Optimize-and-Learn: a Parameter Tuning Framework applied for the Divide-and-Evolve Method in 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

Our method presented in this paper is a surrogate-model based combined learner and optimizer (LaO) for parameter tuning. In this paper LaO is used to tune the our Evolutionary Algorithm based Divide-and-Evolve (Dae) algorithm applied for AI Planning. The LaO framework makes it possible to learn the relation between the features of the instances and the optimal parameters, thus it makes it allows to generalize this relation to unknown instances in the domain or even in a new domain. Moreover, the learned model is already used as a surrogate-model in optimization to improve it. This way we aim to solve intra-domain and extra-domain tuning and generalization in one single framework. Our realization of LaO uses an Artificial Neural Network for learning the mapping and a Covariance Matrix Adaptation Evolution Strategy for optimization. We demonstrate that our algorithm is capable of improving the quality of the DaE algorithm considerably even with only a few iterations. The incorporation of the relation to the ANN model has its limits probably because the feature-set is not rich enough. However, the learned model perform with almost the same quality in the independent tests, which means that the learned model is capable of generalization at least in the domain. Multiple-domain tuning is currently solved by training models per domain, tuning of a general model has not been tried yet.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Artificial Intelligence Problem Solving, Control Methods, and Search

1. 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 (see for instance [29] and [17]), 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 might be as many parameter tuning algorithms as optimization techniques. Several methods have been proposed, for a good review of current state of the art see [18]. Though we have some experience both with Racing ([4]), REVAC ([20]) and ParamILS ([13]), one faces always the same generalization issue: can a parameter-set that has been optimised for a given problem be successfully used to another one? The answer of course depends on the similarity of the two problems. However, even in an optimization domain as precisely defined as AI Planning, there are very few results describing 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 feature-set, 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 [12] 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 has allowed the authors to learn a meaningful mapping between the features and running-time, using function regression. Optimizing this model 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. This difference alone is however not causing major problems. We design the aforementioned framework for any domain, and start with some experimenting. The machine learning techniques used will also be different.

An AI planning task written in PDDL 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 sev-

eral instances sharing the same predicates and actions, but differing either by the number of objects, or the initial and goal states.

Unfortunately, until now, there has not existed any set of features for AI Planning problems in general that was sufficient to describe the characteristics of a planning task, like for the SAT domain [12] mentioned above. In this paper however, we make an important step to the framework for parameter tuning applied generally for each domain in AI Planning 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 optimization phase using the learned model similarly as in other surrogate-model based optimization techniques (see for example [1] for a Gaussian-process-based approach).

LaO can of course be applied to any target algorithm to be tuned, but in this paper we are considering our special, Evolutionary Algorithm (EA) based method, called Divide-and-Evolve (DaE), which is an EA applied to solve AI problems. We take DaE as it is, as a black-box algorithm and have not modified it in this paper. The same realization described in [16] is used.

The paper is organized as follows: AI Planning Problems and the basic YASHP Solver is briefly introduced in section 2. Section 3 describes our intermediate, Divide-and-Evolve Method. Section 4 describes our new, top level method: Learn-and-Optimize used for parameter tuning. Our resuts are presented in section 5 and conclusions are drawn in section 6. We end the paper in section 7 with future work foreseeable.

2. AI PLANNING PROBLEMS

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 the number of actions; it may also be the total cost of the actions with cost, if costs are defined; and it may also be the total makespan for so called durative actions.

Domain-independent planners rely on the Planning Domain Definition Language PDDL2.1 [7]. 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 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, 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 $\mathcal{P}_D(I, G)$.

3. OUR DIVIDE-AND-EVOLVE ALGORITHM

Our Divide-and-Evolve (DaE) approach is fully described in [16], 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 [25], [19], [27], [28], and [6]. 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 [10]. 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 [15].

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 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, but quicker planner, YAHSP is used. YAHSP2 [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 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. 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 [2] 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... [11]). 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 restrict further this list allowing only atoms that are built with a restricted set of predicates. Manual choice had been used in the early versions of DaE [3]. 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 a priori to 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 [11] 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.

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

4. THE LEARN-AND-OPTIMIZE PARAMETER TUNING FRAMEWORK

4.1 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 in parameter tuning, and this is the generality of the tuned parameters. Tuning for one instance has some meaning if only that instance is to be solved. On the other hand, parameters tuned for one instance may not be optimal for other instances, as [5] demonstrates it. This very same paper also demonstrates that global tuning for several domains is even more inferior.

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. For example in [5] per domain tuning was performed with the most difficult, last instance. One can see from the results that these parameters were often suboptimal for the other instances. Probably 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 parameter-setting may be computed.

Inter-domain generalization means that the parameter-setting is generalized for an instance in a different domain (extra-domain generalization). The same problems may arise as in intra-domain generalization, but furthermore, also differences between the domains may cause a problem: Even an

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

instance of similar complexity may require different settings in another domain. The inferior results in [5] with global tuning indicate that these are real problems.

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 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 (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. There is a possibility that different instances have the same features, which may occur if the feature-set is not rich enough. If for two instances the features are the same, we there is a chance 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". We do not know how serious problem this unambiguity could consist. We hope that a learning algorithm would tolerate this and "cut through" the unambiguity. For the sake of simplicity we consider as if there existed an unambiguous mapping from the feature space to the optimal parameter space.

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

.

However, we will indicate, if some problems in the results may be caused by an unambiguity. 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 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 re-training of the learner with the best parameters found.

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.

4.2 Our Implementation of the LaO Framework

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.

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, therefore simply a Covariance Matrix Adaptation Evolution Strategy was chosen, specifically, a (1+1)-CMA-ES (in short, CMA-ES, see [9]). We started CMA-ES with a known set of good default parameters, taken from [5].

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 for a single instance. 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 resemble to a population, but to a set of species. We have to assume that the optimum for the instances are different: they are in different "niche"-s of the parameter tuning "biosphere". However, we also may assume that 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-tranferer was used in our algorithm. When the Genetransferer 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. There is another benefit from gene-transfer, it may smooth out the unambiguities between instances: we increase 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.

Using the ANN and the Genetransferer 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 if they were the hint of the CMA-ES algorithm, i.e. we request a hint from the CMA-ES and change it to be the value of 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 the later case.

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 [8]. 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 to the ANN. Note that the Genetransferer and the ANN itself can not give 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 gene-transfer.

Our realization of our LaO algorithm uses the Shark library ([14]) for CMA-ES and the FANN library for ANN ([21]). To evaluate each parameter-setting with each instance, we use a computer-cluster of LRI 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 evaluations is fixed for DaE. Note that the number of evaluations is approximately proportional with 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, with fixed number of evaluation the number of generations will be limited respectively resulting approximatively in the same running time for each parameter-settings.

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.

5. RESULTS

In the Planning and Learning Part of IPC2011 (IPC), there were 5 sample domains pre-published, with a corresponding problem-generator for each domain. The 5 domains are Ferry, Freecell, Grid, Mprime, and Sokoban. We excluded Ferry from this paper, since there were not enough number of instances to learn a mapping. For the remaining 4 domains we generated approximately 100 instances for each domain, since this seemed to be appropriate for a running time of 2-3 weeks. The description of the track fixes running time as 15 minutes. Since we use number of evaluations as a terminating criterion 11 runs were carried out for each instance on our own, dedicated server to measure the median of number of evaluations with our default parameters. The median of 11 runs were taken and used as a termination criterion for each instance in the train set on any computer. For many instances we did not have any result in 15 minutes, those we had to drop. The remaining instances were used for training.

Table 1 shows the data for each domain, as you can see from the approximately 100 instances from each domain we could use all of the training instances (108) only in the domain Freecell. In the other domains we got much less usable instances. The Mean Square Error (MSE) of the retrained ANN is shown for each domain. Note that since there can be multiple optimal parameters for the same instance (fitness-function is discrete), there might be an unavoidable error of the ANN.

5 iterations of CMA-ES were carried out, followed by one ANN and one Genetransferer, and this cycle was iterated in the algorithm. This means that for example for the Grid domain that LaO was running for 10 iterations and CMA-ES was called 50 times in total. One has to note that this is not much, but we were restricted by time.

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. Over the 10 iterations of LaO in the domain Grid this means that 500 iterations (epochs) of the ANN were carried out in total. However, note that the best parameters were trained with much less iterations, depending on the time when they were found. In the worst case, if the best parameter was found in the last iteration of LaO, it was trained for only 50 and not used anymore, only recorded in the logs.

A parameter in LaO may come from different sources, namely it can be the default parameter, or requested from the CMA-ES, the Genetransferer or the ANN. It is an important information to know how good these sources work in optimization. Table 2 serves this purpose: it shows how each source contributes to the best parameter-settings in the end. For each source the first number shows 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 sources is as it is in the table: for example if CMA-ES found a different parameter-settings with the same makespan as the default, that is not included in the first ratio, but it is included in the second. Analyzing both numbers can lead to interesting conclusions. For example, for domain

Name	numiter	size of training	size of test	ANN-error	Q in LAO	Q of ANN on train	Q of 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
Sokoban	8	32	52	0.11			

Table 1: Domains: note that only the actually usable training instances are shown. numiter=number of LaO iterations. Size of train specifies the number of train instances. ANN-error is given as MSE, returned by FANN. Q=quality-improvement ratio. "In Lao" means best found parameter-set in LaO.

Name	default parameters	CMA-ES	Genetrasnferer	ANN
Freecell	0/0.09	0.64/0.66	0.18/0.08	0.18/0.17
Grid	0.02/0.24	0.66/0.6	0.16/0.11	0.17/0.05
Mprime	0.02/0.45	0.59/0.36	0.2/0.11	0.18/0.08
Sokoban	0.01/0.03	0.69/0.61	0.2/0.32	0.1/0.03

Table 2: Best results according to source of hint. Each cell shows ratios. The first number shows the ratio the corresponding 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.

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. Note that CMA-ES was giving the first hint in each iteration and had 5 times more possibilities than the ANN. Taking this into account both the ANN and Genetransferer made an important contribution to optimization.

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 DAEx are described in table 3. For a detailed description of these parameters, see [5]. 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.

Since testing was also carried out on the cluster, the termination criterion for testing was also the number of evaluations fixed for each instance. For evaluation the quality-improvement (quality-ratio) metric as used as in IPC competitions. As a baseline we took 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. Note that since our termination criterion is number of evaluations, there was no unsolved instance. If an instance was unsolvable, it was dropped, if it was solvable, it is also solvable with any parameter-setting, moreover the running time does not depend on the computer.

Table 1 shows our results. We present several quality-improvement ratios. "In LaO" means that the best found parameter is compared to the default. By definition this ratio can not be less than 1 for any instance. We also present quality-improvement ratios for the retrained ANN on the training-set and the test-set. In these later cases, numbers less then 1 are possible, but were rare. As it can be seen we achieved a considerable quality-gain in training, but the transfer of this improvement to the ANN-model was only partial. Reasons for this may be different. First, there is the unambiguity of the mapping, second, the ANN may not be complex enough for the mapping, but most probably the feature-set is not powerful enough.

On the other hand, the ANN model generalizes excellently to the 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 as for the train set.

Our results are quite similar for each domain. 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 Sokoban instances (32 out of 99 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

Our method presented in this paper is a surrogate-model based combined learner and optimizer for parameter tuning. We demonstrated that our algorithm is capable of improving the quality of the DaE algorithm considerably even with

Name	Minimum	Maximum	Default value
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	100000	10000
Population size	10	300	100
Number of offsprings	100	2000	700

Table 3: Controlled Parameters

only a few iterations. An appropriate number of iterations, like 1000 shall be carried out to demonstrate the capability of the algorithm. We also demonstrated 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.

Naturally, our algorithm also has some parameters, which can be tuned again. Parameter tuning in this respect is 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 with possibly less parameters. But the infinite regress can not be stopped: there will always be some ultimate constants. Similarly, the parameters of any algorithm can be tuned by another algorithm and that may improve the results. But there is no ultimate algorithm. The bad news here is that the computing capacity needed is exploding. Good news is that in each step in the hierarchy an improvement can be made, moreover, the possible improvements become smaller and smaller, thus practically we can stop at some point.

7. FUTURE WORK

The most important experiment to carry out in the future is simply to test the algorithm with more iterations and more domains this takes several month of work even using a 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 apply one of the so-called Growing Neural Network architectures [22].

Also the benefit of gene-transfer and/or cross-over might be investigated further. Gene-transfer shall be improved so that chromosomes are transfered deterministically in order of similarity of instances measured by similarity of features.

One shall also test, how inter-domain generalization works. Maybe it is possible to learn a mapping for all the domains, since the features may grasp the specificity of a domain. Our results indicate that the current feature-set may be too small and it has to be extended for better results. Feature-selection would become important only if the number of features are more than half of the number of examples (well known rule of thumb). Unfortunately, this is not the case yet.

8. ACKNOWLEDGEMENTS

9. RESTRICTIONS ON FURTHER USE

10. 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] 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, *Ninth European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2009)*, number 5482 in Lecture Notes in Computer Science, pages 133–144. Springer-Verlag, 2009.

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

[4] J. Bibai, P. Savéant, M. Schoenauer, and V. Vidal. Learning Divide-and-Evolve Parameter Configurations with Racing. In A. Coles, A. Coles, S. J. Celorrio, S. F. Arregui, and T. de la Rosa, editors, *ICAPS 2009 proceedings of the Workshop on Planning and Learning*, Thessaloniki Grèce, 2009. ICAPS and University of Macedonia, AAAI press.

[5] 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.

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

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

[8] 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.

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

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

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

[18] E. Montero, M.-C. Riff, and B. Neveu. An evaluation of off-line calibration techniques for evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 299–300, New York, NY, USA, 2010. ACM.

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

[20] 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.

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

[22] X. Qiang, G. Cheng, and Z. Wang. An overview of some classical growing neural networks and new developments. In *Proceedings of the 2nd International Conference on Education Technology and Computer (ICETC)*, 2010.

[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.