Learn-and-Optimize: a Parameter Tuning Framework for Evolutionary AI Planning

Brendel Matthias, Marc Schoenauer

Learn-and-Optimize: a Parameter Tuning Framework for Evolutionary AI Planning

Mátyás Brendel[1] and Marc Schoenauer[2]

[1] Projet TAO, INRIA Saclay & LRI
matthias.brendel@lri.fr
[2] Projet TAO, INRIA Saclay & LRI
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 relation to unknown instances in the same domain. Moreover, the learned knowledge is used as a surrogate-model to accelerate the search for the optimal parameters. 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.

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 [25, 14], they usually need computationally expensive parameter tuning. Being a general optimization problem, there are as many parameter tuning algorithms as optimization techniques. However, several specialized methods have been proposed, and the most prominent ones today are Racing [4], REVAC [16], SPO [2], and ParamILS [10]. 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, until now, in AI Planning, sufficiently accurate features have not been specified that would allow to describe the problem, no design of a general learning framework has been proposed, and no general experiments have been carried out. 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 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 similarly, but also in a different way as in standard surrogate-model based techniques (see e.g., [1] for a Gaussian-process-based approach).

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 [13].

The paper is organized as follows: AI Planning Problems and the Divide-and-Evolve algorithm are briefly introduced in section 2. Section 3 introduces the original, top level parameter tuning method, Learn-and-Optimize. The case study presented in Section 4 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 5.

2 AI Planning and Divide-and-Evolve

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.

Domain-independent planners rely on the Planning Domain Definition Language PDDL2.1 [6]. 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, sets the initial state and provides a description of the goal. 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 $\mathcal{P}_D(I, G)$.

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 [20, 15, 22, 23, 5]. 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 [9]. 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 [18, 19]. For a complete formal description, see [12].

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 guarantee of optimality at all calls is not mandatory in order for DaE to obtain good quality results [12], a sub-optimal, but fast planner is used: YAHSP [21] 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.

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.

3 Learn-and-Optimize for Parameter Tuning

3.1 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. Generalizing parameters learned on one instance to another instance might be problematic, because there are instances with very different complexity in the same domain. For example in [3] 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. One workaround to 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. The proposed Learn-and-Optimize framework (LaO) aims at learning such a relation by adding learning to optimization. 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.

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 set $F(\mathcal{I}) \in \mathbf{R}^n$ 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. For the sake of simplicity let us assume that there exists an unambiguous mapping from the feature space to the optimal parameter space.

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

.

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.

The idea of using some surrogate model in optimization is not new. Here, however, there are several instances to optimize, and only one model is available, that maps the feature-space into the parameter-space. A significant conceptual difference is that while in standard surrogate techniques the model is trained and evaluated for fitness, while in our approach we directly evaluate it for optimal parameter candidates.

Nevertheless, there is no question about how to use our model of p in optimization: one can always ask the model for hints about a given parameter-set. 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.

3.2 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 LaO implementation is the choice of the optimizer used for parameter tuning. Because parameter optimization will be done successively for several instances, the simple yet robust (1+1)-Covariance Matrix Adaptation Evolution Strategy [8], in short (1+1)-CMA-ES, was chosen, and used with its robust own default parameters, as advocated in [3].

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 like a crossover between the individuals of this population. Of course, the optimal parameter sets for the different instances are different; However, good 'chromosomes' for one instance may at least help another instance. 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*. This is similar to the migration operator of the so called Island Model Genetic Algorithm [24], and the justification is similar: parallelism and separability. There are however considerable differences: in our case, 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 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.

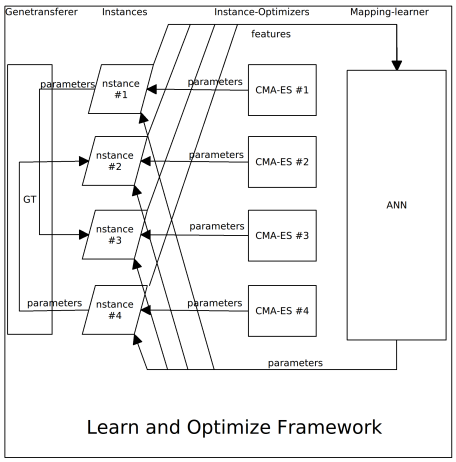

Fig. 1. Flowchart of the Lao framework, displaying only 4 instances.

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 [7], 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 [11] for CMA-ES and the FANN library for ANN [17]. 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. 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. 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.

4 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, approximately 100 instances were generated, with the published generators and distribution (ranges) of generator-parameters 100 instances per domain seemed to be appropriate for a running time of 2-3 weeks. The competition track description fixes running time as 15 minutes. 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.

The real IPC competition domains of the same track were released later. These domains were much harder, meaning that most of the official train instances could not be solved at all by DaE in 15 minutes. Therefore the published instance-generators were used, but with a lower range of the generator-parameters. Even this way, we can only present one domain: Parking. For the other domains, the number of training instances, or the iterations of LaO carried out until the deadline is not sufficient to take the case-study seriously.

Domain Name	# of training instances	# of test instances
Freecell	108	230
Grid	55	124
Mprime	64	152
Parking	101	129

Table 1. Description of the domains

Table 1 presents the train- and testsets for each domain. The Mean Square Error (MSE) of the trained ANN is shown for each domain. Note that because the fitness takes only few values, there can be multiple optimal parameter sets for the same instance, resulting in an unavoidable MSE. So we do not expect this error to converge to 0. One iteration of LaO amounts to 5 iterations of CMA-ES, followed by one ANN training and one Genetransferer. Due to the time constraints, only a few iterations of LaO were run. For example the 10 iterations in domain Grid amounts to 500 CMA-ES calls in total.

The controlled parameters of DaE are described in table 2. For a detailed description of these parameters, see [3]. The feature-set consists of 12 features which are presented in table 3. The first 5 features are computed from the domain file, after the initial grounding of YAHSP: number of fluents, goals, predicates,

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 offsprings	100	2 000	700

Table 2. DaE parameters that are controlled by LaO.

Name	minimum	mean	maximum
# initial fluents	28	31.17	34
# goals	2	2	2
# predicates	7	7	7
# objects	32	35.17	38
# types	3	3	3
mutexdensity	0.14	0.15	0.17
# lines domain	198	198	198
# words domain	640	640	640
# bytes domain	5729	5729	5729
# lines instance	139	153.33	166
# words instance	379	427.42	469
# bytes instance	2017	2265.75	2479

Table 3. Minimum, mean and maximum values are given for the Freecell domain.

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. 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. Note that some features take only one values, they however had meaning when training an inter-domain model.

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, 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.

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 in retraining 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 did the training of the ANN uniquely, i.e. only with the first found best parameters.

Domain Name	# of iterations	ANN error	quality-ratio in LaO	quality-ratio ANN on train	quality-ratio ANN on test
Freecell	16	0.1	1.09	1.05	1.04
Grid	10	0.09	1.09	1.05	1.03
Mprime	8	0.08	1.11	1.05	1.04
Parking	11	0.12	1.49	1.41	1.14

Table 4. 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.

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 obtained by tuning for some representative domains with global racing in previous work see [3]. 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.

$$Q = \frac{Fitness_{baseline}}{Fitness_{tuned}} \tag{2}$$

Table 4 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. 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 original ANN), but were rare.

The first 3 domains in Table 4 have similar results: 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 of the unambiguity of the mapping, or because 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, at least for the first 3 domains. 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 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. The situation is somewhat different for the last domain: Parking. Here we have a high quality-gain in training (almost 50%), even much of this could be learned by the ANN, however, here we have a huge drop for the test-set. The reason for this is that when using the ANN as an extrapolation, there is a considerable number of instances which get unsolvable. Still, we get some overall gain in average, in fact we get the highest gain for this difficult domain. The main issue for such hard domains will be to avoid much more effectively unfeasible parameters when extrapolating the ANN-model for unknown instances.

5 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 over tuning with global racing 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.

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 apply 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. Present results indicate that the current feature set is too small and should be extended for better results. Also a more effective mechanism to avoid unfeasible parameters in the ANN-model has to be developed, especially for hard domains and instances.

6 Acknowledgements

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

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, 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.
4. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. In *GECCO '02*, pages 11–18. Morgan Kaufmann, 2002.
5. A. H. Brié and P. Morignot. Genetic Planning Using Variable Length Chromosomes. In *Proc. ICAPS*, 2005.
6. M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR*, 20:61–124, 2003.
7. 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.
8. N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.
9. W. Hart, N. Krasnogor, and J. Smith, editors. *Recent Advances in Memetic Algorithms*. Studies in Fuzziness and Soft Computing, Vol. 166. Springer Verlag, 2005.
10. 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.

11. C. Igel, T. Glasmachers, and V. Heidrich-Meisner. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.
12. 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.
13. 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.
14. F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, Berlin, 2007.
15. I. Muslea. SINERGY: A Linear Planner Based on Genetic Programming. In *Proc. ECP '97*, pages 312–324. Springer-Verlag, 1997.
16. 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.
17. N. Nissen. Implementation of a Fast Artificial Neural Network Library (FANN). Technical report, Department of Computer Science University of Copenhagen (DIKU), 2003.
18. 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.
19. 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.
20. L. Spector. Genetic Programming and AI Planning Systems. In *Proc. AAAI 94*, pages 1329–1334. AAAI/MIT Press, 1994.
21. 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.
22. C. H. Westerberg and J. Levine. "GenPlan": Combining Genetic Programming and Planning. In M. Garagnani, editor, *19th PLANSIG Workshop*, 2000.
23. 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.
24. D. Whitley, S. Rana, and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1998.
25. T. Yu, L. Davis, C. Baydar, and R. Roy, editors. *Evolutionary Computation in Practice*. Studies in Computational Intelligence 88, Springer Verlag, 2008.