ACOPlan: Planning with Ants

Marco Baioletti, Alfredo Milani, Valentina Poggioni and **Fabio Rossi**
Dipartimento di Matematica e Informatica
University of Perugia, Italy
baioletti,milani,poggioni,rossi@dipmat.unipg.it

Abstract

In this paper an application of the meta–heuristic *Ant Colony Optimization* to optimal planning is presented. It is well known that finding out optimal solutions to planning problem is a very hard computational problem. Approximate methods do not guarantee either optimality or completeness, but it has been proved that in many applications they are able to find very good solutions, often close to optimal ones. Since one of the most performing stochastic method for combinatorial optimization is ACO, we have decided to use this technique to design an algorithm which optimizes plan length in propositional planning. This algorithm has been implemented and some empirical evaluations have been performed. The results obtained are encouraging and show the feasibility of this approach.

Introduction

A planning problem is usually defined as a search problem: finding a solution plan which satisfies the problem constraints. An important improvement of this definition is to consider planning as an optimization problem, with respect to a given metric.

Propositional planning are usually optimized with respect to the plan length or to the number of time steps (if parallel actions are allowed). For instance, HSP, using an admissible heuristic function, is able to provide the former kind of optimization, while many algorithms, including GraphPlan, BlackBox and others, can optimize in the latter form, by using an iterative deepening search method. In (Büttner & Rintanen 2005) a method to find parallel plans with as few actions as possible is described. This method can be encoded as an anytime algorithm.

However the planning framework in which optimization seems to be more natural is planning with *numerical fluents*. Actions can handle and modify numerical variables and an objective function can be easily expressed in terms of some target variables (for instance fuel as in *Logistics* domain). However, there exist some other planning models in which an objective function is significative, e.g. the so called *makespan* in temporal planning, the probability that a plan reached the goals in probabilistic planning, the *plan quality* in planning with preferences, etc. Besides the apparent importance of optimization in planning, this topic has not been so extensively studied. Apart from propositional planning, there exist only a few optimal planners (often not so efficient) and there are a relatively small number of other planners which usually produce good solutions, without guaranteeing the optimality.

The main idea of this paper is to use the well known *Ant Colony Optimization* meta–heuristic (henceforth *ACO*) to optimal planning. ACO is a meta–heuristic inspired by the behaviour of natural ants colony which has been successfully applied to many *Combinatorial Optimization* problems. Clearly, being ACO a stochastic algorithm, there is no guarantee that optimal solutions are ever found but we hope that, as shown in many other applications of ACO, this method can produce excellent or optimal solutions.

To test if ACO can be effectively used in optimization planning problem we have implemented, as a first step, an ACO–based propositional planner which tries to optimize plans in terms of plan length. Since we will show that this implementation is able to find very good solutions, it is likely that ACO can be successfully implemented in other forms of planning, as we will discuss in the conclusions.

As far as we know, this is the first application of ACO to optimal automated planning. Evolutionary techniques, like *Genetic programming* in (Muslea 1997), have been sometimes used in planning, but without good results.

The paper is structured as follows. A short introduction to Ant Colony Optimization is provided, then we describe the details of the application of ACO to AI Planning. We then discuss about some planning systems which can be directly related to this work. Some experimental tests are described and their results are discussed. The paper is concluded by describing some possible improvements and extensions of this work.

Ant Colony Optimization

Ant Colony Optimization (ACO) is a meta–heuristic born to tackle Combinatorial Optimization problems introduced since early 90s by Dorigo et al. (Dorigo 1992; Dorigo, Maniezzo, & Colorni 1996; Dorigo & Stuetzle 2004). ACO is inspired by the foraging behavior of natural ant colonies. When walking, natural ants leave on the ground a chemical substance called *pheromone* that other ants can smell.

Choosing its way one ant tends to probabilistically choose paths marked with a stronger quantity of pheromone. When an ant finds some food it comes back to the nest and it releases a higher quantity of pheromone. Next ants leaving the nest will tend to follow this stronger pheromone trail and to release themselves further pheromone on. As time goes by, the number of ants that follow trails marked by a higher quantity of pheromone increases and favorite trails arise.
The ants' behaviour has inspired the design of Ant Colony Optimization (ACO) algorithms. They can be used to solve combinatorial optimization problems whose solutions are defined in terms of sequences of components. A component is an edge in the so called *construction graph* (Dorigo & Blum 2005) $C_g = (V, E)$, while a solution is a path connecting the initial node to a final node. A problem can specify constraint on which components can be chosen according to the current node and possibly the previous components. A classical basic framework for ACO works as follows. At each iteration, each of m ants builds a solution adding n components one by one. Each solution component is assigned a real value, the so called *pheromone value*. The next component to add is probabilistically drawn from the set of the admissible components (i.e. the components that satisfy the problem constraints). Most ACO algorithms (Dorigo & Stuetzle 2004) choose the components according the following distribution (called *transition probabilities*):

$$p(c) = \frac{[\tau(c)]^\alpha \cdot [\eta(c)]^\beta}{\sum_{c' \in \Re(s^p)} [\tau(c')]^\alpha \cdot [\eta(c')]^\beta}, \quad \forall c \in \Re(s^p) \qquad (1)$$

where s^p is the current partial solution, $\Re(s^p)$ the set of feasible solution components (it is determined at each addition step according to the problem constraints), τ is the value of pheromone for the components, η is a heuristic function that evaluates how promising a component is, and α and β determine the relative importance of pheromone value and heuristic estimation. When all the m solutions have been built, a local search method can be optionally applied to improve them. Then a pheromone update is performed according to the quality of solutions found, i.e. components constituting best solutions receive a higher quantity of pheromone, changing the probability of each component to be drawn at next iteration. Many ACO variants differ just on the pheromone update method adopted: the solutions set involved varies, the rule for updating differs, etc. Often the best solutions considered for the update are *best–so–far* (best solution found so far) or *iteration–best* (best solution found in the current iteration) or both. In certain variants limits for maximum and minimum pheromone values are used. Most ACO algorithms use the following update rule (Blum 2005): for each component c

$$\tau(c) = (1 - \rho) \cdot \tau(c) + \rho \sum_{s \in \Psi_{upd} \mid c \in s} F(s) \qquad (2)$$

where $\rho \in (0, 1]$ is the evaporation rate, Ψ_{upd} is the set of solutions involved in the update and F is the so called *quality function* which evaluates the goodness of the solution[1]. The evaporation rate decreases the pheromone values to avoid a premature convergence towards non optimal solutions. This process is repeated a predefined number of iterations or until a good enough solution is found.

[1] F is cleary related to the function to be optimized

ACO and Planning

At a first sight the implementation of an ACO algorithm for the problem of optimizing propositional planning (in terms of plan length) seems straightforward because of the apparent similarity between the construction of solutions component by component in ACO and the forward search in the state space in planning.

Therefore we use a colony of m ants, each of them builds up a plan starting from the initial state $s_0 = \mathcal{I}$ and executing actions step by step. Ants draw the next action to execute in the current state s from the set of executable actions (i.e. actions having their preconditions satisfied in s). After executing a the current state is updated as $s' = Res(s, a)$, being $Res(s, a) = s \cup add(a) \setminus del(a)$. Each ant adds components until the maximum allowed plan length is reached, or a solution plan is found or no action can be executed.

Many features must be decided to fully implement an ACO algorithm, so in this section we describe our proposal for the most important ones.

Pheromone models

Besides the fact that a solution is composed by actions, this is a convincing argument that for the pheromone model a *solution component* cannot be just an action. In fact, the same action a could be executable in different states leading to different successor states, but its usefulness might be different depending on the state in which it is performed. If a takes part of a solution used for the pheromone updating, the contribution given by the pheromone to the transition probability of a is increased, independently on the current state and therefore even in states in which a is harmful.

A first possibility is to assume the couple state/action $c = (s, a)$ as a *solution component*. In this way the pheromone part takes into account the state in which a is executed and gives a larger contribution only for those states in which a is supposed to be useful. This pheromone model has been called *state–action* model. A nice feature of this model is that the construction graph coincides with the state–space graph, where the set of vertices V is the set of possible states and the edges E correspond to transitions. A drawback of this model is that the number of components can grow very fast. To overcome this problem an efficient data structure is used to store, update and retrieve the pheromone values.

Another possibility, which has a much smaller number of components, is to assume the couple time step/action $c = (t, a)$ as a *solution component*, where t is the time step at which a is executed. This pheromone has been called *level–action* model. Intuitively this model gives a less useful information if compared to the information provided by the previous model. In fact, while the pheromone values of first model can be easily explained (the convenience of

executing an action depends on the context), the second model relates the convenience of an action to the time in which is executed, no matters in which context is executed.

A third possibility is to evaluate only the states. In this pheromone model, called *state* model, a component is just a state. In this model the information provided by the pheromone is how convenient is to pass through a given state.

Referring to the *state–action* pheromone model the rule to calculate the *transition probabilities* in the state s is the following

$$p(a|s) = \frac{[\tau(s,a)]^\alpha \cdot [\eta(Res(s,a))]^\beta}{\sum_{a' \in \mathcal{A}(s)} [\tau(s,a')]^\alpha \cdot [\eta(Res(s,a))]^\beta}, \quad \forall a \in \mathcal{A}(s) \tag{3}$$

where $\tau(s,a)$ is the pheromone value assigned to component (s,a), $\eta(s')$ is a heuristic function that evaluates how promising is to reach the state s', $\mathcal{A}(s)$ the set of actions executable in s. Finally, α and β are parameters to determine the relative importance of pheromone value and heuristic estimation.

Similar formulae are used to compute the transition probabilities for the other two pheromone models.

Heuristic Estimation η

We decided to choose as heuristic function η that one used in *Fast-Forward (FF)* (Hoffmann & Nebel 2001). *FF* estimates the distance from a state s to the goals, i.e. the number of actions needed to reach the goals. *FF* exploits the basic idea of relaxing the original planning problem ignoring deleting effects of all actions, introduced also in other heuristic system (for instance *HSP* (Bonet & Geffner 2001)). Initially *FF* builds a relaxed planning graph (ignoring delete effects) starting from a given state. The graph is extended until goals are reached. Then it attempts to extract a relaxed plan in a *GraphPlan* style. The number of actions spent $h(s)$ is the estimated distance. Moreover *FF* provides some pruning techniques also, in order to exclude some space state branches from search. Since FF is able to compute, with a negligible amount of time, the so called *Helpful Actions* (actions that seem more promising than other ones) we decide to increment the η value for these actions.

The value $\eta(s)$ needed to compute the transition probabilites is then obtained by means of the following formula:

$$\eta(s) = \begin{cases} \dfrac{1}{h(s)(1-k)} & \text{if } s \text{ is reached by a Helpful Action} \\ \dfrac{1}{h(s)} & \text{otherwise} \end{cases} \tag{4}$$

where k is the reduction rate (usually in our tests we set $0.15 \leq k \leq 0.5$) to increase the transition probabilities of *Helpful Actions*.

Plan Evaluation

At the end of each iteration a quality evaluation of all plans built by the ants is needed to perform a pheromone update. An intuitive (trivial) criterion is to consider the number of goals reached, but this is useless when no plans reach any goal.
The basic idea to evaluate the quality of a plan is keeping track of h_{min} and t_{min}, where h_{min} is the minimum heuristic value (i.e. minimum distance from goals) reached during plan execution and t_{min} is the first time step in which h_{min} is achieved. Then, a plan π is better than a plan π' if $h_{min}(\pi) < h_{min}(\pi')$ or if $h_{min}(\pi) = h_{min}(\pi')$ and $t_{min}(\pi) < t_{min}(\pi')$.
A *quantitative* measure $Q(p)$ for the quality of plan p can be easily defined as

$$Q(p) = \left(\frac{1}{1+h_{min}}\right)^\gamma \left(\frac{1}{t_{min}}\right)^\delta \left(1 + \frac{g_{found}}{g_{count}}\right)^\theta \tag{5}$$

where γ, δ and θ are parameters to tune the importance of three terms, g_{found} is the number of goals reached at construction step t_{min} and g_{count} is the total number of goals. In the experimental test we used $\gamma = \delta = 1$ and $\theta = 0$.

Pheromone Updating

We have decided to perform a pheromone update considering *best–so–far* and *iteration–best* solutions. Referring to the *state–action* pheromone model the rule is the classical one used in the HyperCube Framework for ACO (Blum, Roli, & Dorigo 2001):

$$\tau(s,a) \leftarrow (1-\rho)\tau(s,a) + \rho \sum_{\pi \in P_{upd} \,|\, (s,a) \in \pi} \frac{Q(\pi)}{\sum_{\pi' \in P_{upd}} Q(\pi')} \tag{6}$$

where ρ the pheromone evaporation rate, P_{upd} is the set of solutions (plans) involved in the update and Q is the quality plan evaluation function.
Note that in this formula we have decided, after some preliminary evaluation, to update, for each $\pi \in P_{upd}$, the pheromone values relative only to the first t_{min} actions. In this way the part of a plan after having reached the best state (in terms of heuristic function) is considered useless (it reaches only worst or equivalent states) and then is neglected.
The pseudo code of the resulting algorithm is shown in figure 1.

Related Works

There are several relevant planners which can be directly related to this work. First of all we have to cite LPG (Gerevini & Serina 2002) because it is based on a stochastic algorithm. It is important to note that stochastic approaches to planning begins to receive the right attention by the planning community (with respect to the standard deterministic approaches) because it has been proved they can yield excellent results also in the case of optimality problems. Moreover, we have to cite heuristic planners like HSP (Bonet & Geffner 2001) and FF (Hoffmann & Nebel 2001) for two different reasons:

Algorithm 1 The algorithm ACOPlan

$s_{best} \leftarrow \emptyset$
InitPheromone(T, c)
for $g \leftarrow 1$ to *number of generations* **do**
  $s_{iter} \leftarrow \emptyset$
  for $m \leftarrow 1$ to *number of ants* **do**
    $s_p \leftarrow \emptyset$
    $state \leftarrow$initial state of the problem
    for $i \leftarrow 1$ to *max length* **do**
      $A_i \leftarrow$ feasible actions on $state$
      $H_i \leftarrow \emptyset$
      $HA_i \leftarrow$ *GetHelpfulActions*($state$, A_i)
      for all a_i^j in A_i **do**
        $h_i^j \leftarrow$ heuristic value of a_i^l
        $H_i \leftarrow H_i \cup h_i^j$
      end for
      $a_k \leftarrow$ *ChooseAnAction*(T, H_i, A_i, HA_i)
      extend s_p adding a_k
      update $state$
    end for
    if $f(s_p) > f(s_{iter})$ **then**
      $s_{iter} \leftarrow s_p$
    end if
  end for
  if $f(s_{iter}) > f(s_{best})$ **then**
    $s_{best} \leftarrow s_{iter}$
  end if
  UpdatePheromone(T, s_{best}, s_{iter}, ρ)
end for

(i) the heuristic of our planner is directly inspired from the FF's one, *(ii)* the HSP planner can run in an optimal version and its results can be used also to compare the solution plans given by the planners. Finally, some words have to be spent about the optimality concept in planning. The notion of optimal plan is first introduced as makespan (both as number of actions and number of steps) and then it has been refined to consider any metric which can also include resources, time and particular preferences or time trajectory constraints. To the best of our knowledge the optimal version of HSP is the best optimal planner with respect the number of actions, so we have included it in our experimental tests. It is important to note that we have not included optimal planners like SATPLAN (Kautz & Selman 1999) because, in this case, the optimality is expressed in terms of number of planning level and the results are not comparable.

Experimental results

ACOplan has been tested over some domains taken from the International Planning Competitions (IPC). In general these domains are used as standard benchmarks to compare planner performances. We run a set of systematics tests over the domains *Rovers* and *Depots*, *Blocksworld*, *Driverlog*. They have been chosen among the set of benchmark domains because they offer a good variety and the corresponding results allow us interesting comments. For lack of space only results for *Rovers* and *Driverlog* are reported. The results for

	ACOplan		FF		LPG	
Problem	length	time	length	time	length	time
1	10	0.04	10	0	10	0.04
2	8	0.03	8	0	8	0.01
3	11	0.44	13	0	11	0.71
4	8	0.06	8	0	8	0.02
5	22	0.4	22	0	22	0.04
6	36	43.71	38	0	36	2.32
7	18	0.47	18	0	18	10.86
8	26	1.8	28	0	26.1	158.48
9	31	173.67	33	0	31	0.09
10	35	85.37	37	0	35	133.03
11	30.54	2926.36	37	0	30	13.41
12	19	1.14	19	0	19	0.17
13	44	294.19	46	0.02	43.65	190.79
14	28	74.94	28	0	28	0.94
15	41	7459.52	42	0.01	42.25	474.17
16	41	1155.04	46	0.02	41	116.97
17	47.13	3463.33	49	0.02	47	10.33
18	41	5518.31	42	0.04	41.85	434.882
19	66.25	6017.49	74	0.21	68.7	519.622
20	97.5	5189.75	96	0.54	92	480.209

Table 2: Results for *Rovers* domain collecting solution lengths and CPU time

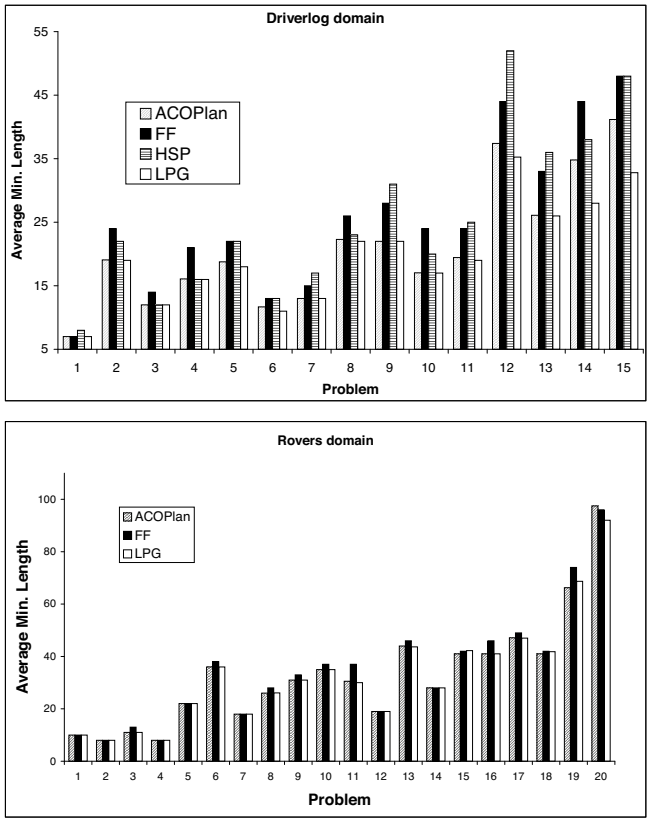

Figure 1: Average Minimum Length found for – *Driverlog* and – *Rovers* domains

	ACOplan		HSP		FF		HSP -opt		LPG	
Problem	length	time	length	time	length	time	length	time	length	time
1	7	0.02	7	0.01	8	0.01	7	0.01	7	0.03
2	19.07	11.18	24	0.01	22	0.01	19	3905.92	19	0.53
3	12	0.09	14	0.01	12	0.01	12	0.14	12	0.08
4	16.07	57.79	21	0.01	16	0.01	16	5854.79	16	0.08
5	18.77	62.37	22	0.01	22	0.01	–	–	18	2.28
6	11.67	71.93	13	0.01	13	0.01	11	0.72	11	0.85
7	13	1.64	15	0.01	17	0.01	13	5796.76	13	0.08
8	22.3	312.94	26	0.01	23	0.01	–	–	22	9.25
9	22	17.96	28	0.03	31	0.01	–	–	22	0.54
10	17.03	116.44	24	0.03	20	0.01	–	–	17	0.54
11	19.43	193.46	24	0.07	25	0.01	–	–	19	23.06
12	37.4	2080.89	44	0.07	52	0.22	–	–	35.25	237.28
13	26.1	750.67	33	0.16	36	0.09	–	–	26	4.55
14	34.8	3301.83	44	1.66	38	0.12	–	–	28	12.54
15	41.17	3454.29	48	1.65	48	0.03	–	–	32.8	499.78

Table 1: Results for *Driverlog* domain collecting solution lengths and CPU time

the other domains are similar.
We chose to compare ACOplan with LPG, HSP and FF.
LPG is very performant and, when run with the *b*est quality (option *-n*), it gives solution plans with, in general, a number of actions very close to the optimum (sometimes it can find solutions with the optimum number of actions). It is a non deterministic planner, so the results collected here are the mean values obtained over 100 runs.
HSP can run with several options. In particular using an A*–like algorithm and a admissible heuristic (options *-d backward*, *-h h2max* and *-w 1*) it produces optimal plan in the number of actions. Nevertheless, in this setting, it often fails to find a solution because it runs out of memory; for this reason we have chosen to run it also with default options in order to solve a larger set of problems and collect more results.
FF has no option to choose and it runs in default version.
ACOplan has many parameters that have to be chosen. We have preliminarly performed a first set of experiments in which we have looked for the best values of α, β and ρ. The experiments were run on five problems of *R*overs domain and five problems of *Z*enoTravel domain. The chosen values for α and β have been $\{1, 2, 5, 7\}$, which are the most used values in ACO literature, while ρ have been varied in $\{0.1, 0.15, 0.2\}$. Table 3 and Table 4 show how many times each value of the parameters appears in the best or in the second best combination. According to these results, we decided to use this setting: 10 ants, 5000 iterations, $\alpha = 1$, $\beta = 7$, $\rho = 0.15$, $c = 1$, $k = 0.5$, pheromone model state–action. Being a non deterministic system, like LPG, the results collected here are the mean values obtained over 100 runs.
In Table 1 and Table 2 results of tests over *Driverlog* and *Rovers* domains are shown. In the first column problem numbers are listed; in the next columns the length of solution plans and execution times are reported for each planner; the column entitled *HSP -opt* contains the results for HSP called with options guaranteeing the optimality. The symbol – in table entries means that the corresponding problem has not

	Rovers										
	Alpha				Beta				Rho		
	1	**2**	**5**	**7**	**1**	**2**	**5**	**7**	**0.1**	**0.15**	**0.2**
Best	1	3						4	2		2
Second	4							4	1	1	2

Table 3: Tuning *Rovers*: best performing parameters

be solved in 2 hours of CPU times or because of memory fault.
Results in Table 1 for the *Driverlog* domain show how the quality of solutions synthesized by ACOplan is practically always better than the ones extracted by FF and HSP and is very close to the ones extracted by LPG. For instance, with respect to FF, on the average, the percentage improvement is 15%, with a top 31%. Moreover, the available data for the optimal version of HSP show how the length of the solution extracted by ACOplan is actually the optimum length.

Results in Table 2 for the *Rovers* domain show similar results where the percentage improvement is 8% with respect to FF. In one case ACOPlan has found in average better solutions than LPG, while in the other cases the average lengths found by the two planners are again very close.

Nevertheless we have obtained good results from an optimality point of view, the same cannot be said about efficiency. Anyway this is not surprising because we have still a quite simple implementation; on the contrary the number of solved problems with respect to the optimal HSP is encouraging and a dramatic improvement of performances is predictable.

Conclusions and Future Works

In this paper we have described a first application of the Ant Colony Optimization meta–heuristic to Optimal Propositional Planning. The preliminary empirical tests have shown encouraging results and that this approach is a

	Zenotravel										
	Alpha				Beta				Rho		
	1	**2**	**5**	**7**	**1**	**2**	**5**	**7**	**0.1**	**0.15**	**0.2**
Best	3	1						4	3	1	
Second	3	1						4	1	2	1

Table 4: Tuning *Zenotravel*: best performing parameters

viable method for optimization in classical planning. For these reasons we are thinking to improve and extend this work in several directions.

First of all, we have planned to modify the implementation of the ACO system, in particular the use of heuristic functions which require a smaller computation time. Hence it is possible that a less informative and less expensive heuristic function can be used without having a sensitive loss of performance.

Then, another idea is to change the direction of the search in the state space: using "regressing" ants, which start from the goal and try to reach the initial state. Backward search methods has been successfully used in planning.

Finally we are considering to apply ACO techniques also to other types of planning. The extension of classical planning which appears to be appealing for ACO is planning with numerical fluents: in this framework an objective function can be easily defined. It is almost straightforward to extend our ACO system (with "forward" ants) in order to handle the numerical part of a planning problem, even if it could be problematic to use the complete state in the solution components. In this case a suitable pheromone model could be the "level–action" model. Also the extension to handle preferences seems to be straightforward, being necessary only a modification in the computation of $Q(p)$.

Acknowledgements

We acknowledge the usage of computers at IRIDIA (http://code.ulb.ac.be/iridia.home.php) Institute at *Université Libre de Bruxelles*, which Fabio Rossi was visiting from June to August 2008, for some of the computations done here. Moreover, a special thank to Dr. Thomas Stützle for his useful suggestions.

References

Blum, C.; Roli, A.; and Dorigo, M. 2001. The hyper-cube framework for ant colony optimization. In *IEEE Transactions on Systems, Man, and CyberneticsPart B*, 399–403.

Blum, C. 2005. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews* 2(4):353–373.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129((1-2)).

Büttner, M., and Rintanen, J. 2005. Satisfiability planning with constraints on the number of actions. In *ICAPS 2005*, 292–299.

Dorigo, M., and Blum, C. 2005. Ant colony optimization theory: a survey. *Theor. Comput. Sci.* 344(2-3):243–278.

Dorigo, M., and Stuetzle, T. 2004. *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press,.

Dorigo, M.; Maniezzo, V.; and Colorni, A. 1996. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics* 26(1):29–41.

Dorigo, M. 1992. *Optimization, learning and natural algorithms*. Ph.D. Dissertation, Politecnico di Milano.

Gerevini, A., and Serina, I. 2002. LPG: a planner based on local search for planning graphs. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02), AAAI Press, Toulouse, France*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253 – 302.

Kautz, H., and Selman, B. 1999. Unifying sat–based and graph-based planning. In *Proceedings of IJCAI-99, Stockholm*.

Muslea, I. 1997. Sinergy: A linear planner based on genetic programming. In *ECP 97*, 312–324.