An ACO approach to planning

No Author Given

No Institute Given

Abstract. In this paper we describe a first attempt to solve planning problems through an Ant Colony Optimization approach. We have implemented an ACO–inspired algorithm that performs a meta–heuristic search to find optimal solutions of a propositional planning problem with respect to the plan length. Since planning is a hard computational problem, stochastic methods are suitable to find good solutions in a reasonable computation time. Preliminary experiments are very encouraging because the system often find better solutions than state of art planning systems. Moreover this framework seems to be easily extensible to other planning models.

1 Introduction

Automated planning [1] is a very important research field in Artificial Intelligence. Planning has been extensively and deeply studied and now many applications of automated planning exist, ranging from robotics to manufacturing and to interplanetary missions.

A planning problem can be concisely described as the problem of finding a plan, i.e. a sequence of actions, which starting from a known initial state leads to a final state that satisfies a given goal condition. Each action can be executed possibly only in some states and in this case, after its execution, the current state is altered by changing the value of some state variables.

Usually planning problems are defined in terms of pure search problem, in which the aim is to find any solution of the problem. Sometimes just finding a solution is satisfactory. But there are many other problems having several solutions and stopping the search phase just after having found a first plan may not be a good idea, because this plan may be sensitively more expensive than other possible plans. The cost of a plan is an important feature to take into account because high cost plans can be useless, almost like non executable plans.

Therefore in these situations the purpose of search is finding optimal or near–optimal plans in terms of a given objective function.

Perhaps the problems in which the role of an objective function to be optimized is more natural are those belonging to the numerical extension of planning. In this framework planning problems, which are usually described in terms of propositional variables, can also contain numerical variables in preconditions, effects, initial state, goal and, of course, objective function.

There exist some other planning models in which an objective function is significative.
In the temporal planning, actions can have different durations and the objective function is the so called makespan (i.e. the duration of the entire plan). In probabilistic planning, where actions can have stochastic effects and the initial state can be not exactly known, the objective function is the probability that the plan reaches the goals. In other terms this function measures how likely the plan solves the problem.

Finally, in the framework of planning with preferences, an objective function can be defined in terms of violations of soft constraints. A soft constraint is a constraint (for instance a precondition or a goal) which is not required to be satisfied, but if it is, the resulting plan is better. In this model the objective function measures the quality of the solution plan.

But even in the case of pure propositional planning, also the number of actions (possibly weighted) is an interesting objective function.

Besides the apparent importance of optimization in planning, this topic has not been extensively studied. In fact there exist only a few optimal planners and there are a relatively small number of other planners which usually produce good solutions, without guaranteeing the optimality.

One reason is that planning is a very hard task even in the pure search version and looking for optimal solutions can only be harder. There are even planning domains, like the blocks world, in which a solution could be found in polynomial time, but searching for the shortest solution is an NP-complete problem.

In any case, the problem of optimal planning is a (hard) combinatorial optimization problem, for which there exist only a few standard algorithmical techniques. Usually these methods are not particularly efficient and are able to solve only small instances of interesting planning problems.

To solve this difficulty, the main idea of this paper is to use and adapt the well known Ant Colony Optimization meta-heuristic to optimal planning. Clearly, being ACO a stochastic algorithm, there is no guarantee that the optimal solution is found but we hope that, as shown in many other applications of ACO, this method produces excellent or optimal solutions.

ACO has been successfully used in many combinatorial optimization problems [2], being competitive with the best ad-hoc algorithms for these problems.

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 in the tests we have performed, 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 an evolutionary meta-heuristics to optimal automated planning. Genetic programming has been used in planning in [3], but this technique has not been shown to be promising and seems to have been soon abandoned.

To adapt ACO to planning we decided to perform a forward search in the state space in which each ant builds up an executable sequence of actions. Ants are guided by the pheromone values and a heuristic function typically used in plan-
One of the most difficult problems we have coped with is how to assign a meaningful score to plans which are not solutions, being undefined the objective function. It is obvious that these situations appear often during the first generations and it is very important to force the ants to produce legal plans as much as possible. Therefore we decided to use a further score function which takes into account the distance between the states reached by the sequence and the goals. Discarding illegal solutions or just counting the number of goals reached would be too naive techniques which will lose many important information about the planning domain.

The paper is structured as follows. In section 2 a short introduction to AI Planning is provided, while in Section 3 we show how we applied ACO to AI Planning. In Section 4 some related works are discussed. In section 5 some preliminary experimental results are shown. Section 6 concludes the paper by describing some possible improvements and extensions of this work.

2 A brief introduction to Automated Planning

The standard reference model for planning is the Propositional STRIPS model [1], called also “Classical Planning”.

In this model the world states are described in terms of a finite set $F$ of propositional variables: a state $s$ is represented with a subset of $F$, containing all the variables which are true in $s$.

A Planning Problem is a triple $(I, G, A)$, in which $I$ is the initial state, $G$ denotes the goal states, and $A$ is a finite set of actions. An action $a \in A$ is described by a triple $(pre(a), add(a), del(a))$. $pre(a) \subseteq F$ is the set of preconditions: $a$ is executable in a state $s$ if and only if $pre(a) \subseteq s$. $add(a), del(a) \subseteq F$ are respectively the sets of positive (negative) effects: if an action $a$ is executable in a state $s$, the state resulting after its execution is $Res(s, a) = s \cup add(a) \setminus del(a)$. Otherwise $Res(s, a)$ is undefined.

A linear sequence of actions $(a_1, a_2, \ldots, a_n)$ is a plan for a problem $(I, G, A)$ if $a_1$ is executable in $I$, $a_2$ is executable in $s_1 = Res(I, a_1)$, $a_3$ is executable in $s_2 = Res(s_1, a_2)$, and so on. A plan $(a_1, a_2, \ldots, a_n)$ is a solution for $(I, G, A)$ if $G \subseteq s_n$.

Usually a planning problem is stated as a pure search problem, in which the purpose is finding, if any, a solution plan. This computational problem is known to be PSPACE–complete.

There exist many algorithmic solutions for planning. Among all the proposed algorithms, three approaches are important.

A first approach is to design ad–hoc algorithms. Until mid 90s the dominating technique has been Partial Order Planning. A more recent way is the graph–based approach, also used in [4]. A second approach is to translate a planning problem into a different combinatorial problem, like propositional satisfiability [5], integer programming or constraint satisfaction problems, and to use fast solvers designed for these problems. A third possibility is to formulate planning
problems as heuristic search problems, as done in HSP [6] and in FF [7].

The Classical planning model has been extended in many directions. Among them it is worth to mention the introduction of a numerical counterpart in planning problems: states are also described in terms of numerical variables and actions can cite numerical variables in preconditions and change the value of numerical variables as effect.

Other interesting extensions are probabilistic planning, in which states are probability distributions and actions can have a stochastic effects, so plans could reach the goal only with a certain probability, and planning with preferences, where some constraints (preconditions and goals, for instance) can also be neglected, reducing hence the plan quality.

3 ACO and Planning

One of most important methods proposed to solve planning problems is the heuristic search in the state space. The search moves from the initial state through state space using well known search methods ($A^*$, Hill climbing, etc.), guided by a heuristic function and stops when a state containing the goals is reached. Usually these methods are deterministic.

In our approach a stochastically meta–heuristic search in the state space is proposed, using an Ant Colony Optimization approach. This is an approximate method, so neither optimality nor completeness is guaranteed. In our approach we are interested on to optimize some quantitative characteristics of a planning problem, as length of solutions found, consumption of resources, and so on. In other words we are regarding to a planning problem as an optimization problem, not only a search problem.

3.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a meta–heuristic designed to tackle Combinatorial Optimization problems and introduced since early 90s by Dorigo et al. [8, 2].

ACO is a constructive meta–heuristic, i.e. artificial ants build up sequences of elements, called components, in an incremental stochastic way. Each solution component is assigned a pheromone value, i.e. a numerical real quantity. The vector of all pheromone values $T$ is the pheromone model. At each construction step the next component to add to the current partial solution is chosen according to a probability distribution on the feasible components (i.e. components that satisfy problem constraints), induced by the pheromone values and a heuristic function that estimates how promising each component is. Ants iterate such a process construction and at end of each iteration the pheromone model is updated increasing the pheromone values of components taking part of better solutions found. In this way the probability of each component to be drawn in the next iterations changes. As time goes by, ants learn which components take
part of better solutions because their pheromone values tend to increase. Many ACO variants differ just on the pheromone update method adopted: the solutions set involved, the rule for updating, etc. Often the solutions considered for the update are best-so-far (best solution found in all the iterations already executed) or iteration-best (best solution found in the current iteration) or both. In certain variants, limits for maximum and minimum pheromone values are used [9]. Another important ACO feature is the so called forgetting mechanism: at each iteration, before charging pheromone values of component taking part of better solutions, all the values are decreased by an evaporation rate $\rho$, in order to prevent a premature convergence to suboptimal solutions.

For more details on ACO see [10, 2].

### 3.2 Planner Ants

In our approach ants build up plans starting from the initial state $s_0 = I$ and executing actions step by step. Each ant chooses the next action $a$ to execute in the current state $s_j$ by drawing from the set $A(s_j)$ of all actions executable in $S_j$. After the execution of an action $a$ the current state is updated as $s_{j+1} = Res(s_j, a)$

Apparently we might assume a single action as a solution component. But the same action $a$ may be executed in different states leading to different successor states. In other words the execution of an action $a$ can have a different utility depending on the state in which is executed.

Consider for example the domain Gripper where a robot has to move the balls $(B1, B2)$ from the room $R1$ to the room $R2$. The initial part of the related state space graph is shown in Fig. 1.

![Fig. 1. Initial part of state space graph for the Gripper domain](image_url)
Action move(R1, R2) may be a useful action or not depending on the state in which it is executed. For example executing this action in s0 is not useful to reach the goals, instead executing this action in s1 is useful. So, if we assume a single action as solution components, charging the pheromone value of move(R1, R2) increases its probability to be drawn in next iterations in both s1 (where it is desirable) and in s0 (where it is not desirable). This consideration leads to assume a couple state/action \(c^j_i = (s_j, a_i)\) as a solution component. In this way charging the component (s1, move(R1, R2)) increases the probability of drawing this action only when it is useful. The whole set of couples state/action constitutes the pheromone model that we have called state–action. In this model the state space graph is the construction graph \(C_g = (V, E)\) \(^1\), where the set of vertices \(V\) is the set of possible states and the set of edges \(E\) is constituted by the actions executable in each vertex.

Another possibility is to assume a couple step/action \(c^j_i = (t_j, a_i)\) as a solution component, where \(t_j\) is the time step at which \(a_i\) is executed. However knowing that an action \(a\) is executed at step \(t_j\) instead of step \(t_j + 1\) seems intuitively less interesting than knowing in which state an action is executed. The whole set of couples step/action constitutes the pheromone model that we have called level–action.

Referring to pheromone model state–action the rule to calculate the transition probabilities is the classical one:

\[
p(a_i|s_j) = \frac{[\tau^j_i]^\alpha [\eta(a^j_i)]^\beta}{\sum_{a_k \in N(s_j)} [\tau^j_k]^\alpha [\eta(a^j_k)]^\beta}, \quad \forall a_i \in A(s_j) \tag{1}
\]

where \(\tau^j_i\) is the pheromone value assigned to component \(c^j_i = (s_j, a_i)\), \(\eta(a^j_i)\) is a heuristic function that evaluates how much is promising to execute \(a_i\) in the state \(s_j\), and \(\alpha, \beta\) are parameters to determine the relative importance of pheromone value and heuristic estimation.

### 3.3 Heuristic Estimation \(\eta\)

We decided to use as function \(\eta\) the heuristic Fast-Forward (FF) \([7]\). FF estimates the distance from a state 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 [6]). 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, i.e. performing a backward search from goals to initial state. The number of actions spent is the estimated distance. Moreover FF provides some pruning techniques also, in order to exclude some space state branches from search. We

\(^1\) In an ACO algorithm the so called construction graph is the graph where ants move over
adopted the so called *Helpful Actions* (actions that seem more promising than other ones) method to increment the $\eta$ value for these actions. So at each construction step, our algorithm calculates $\eta$ values of states resulting by the execution of each feasible action in the current state using the following rule:

$$
\eta(a_i, s_c) = \begin{cases} 
1 & \text{if } a_i \text{ is a Helpful Action} \\
\frac{1}{h(a_i, s_c)(1-k)} & \text{otherwise} \\
\end{cases} \quad \forall a_i \in A(s_c) \tag{2}
$$

where $s_c$ is the current state, $h(a_i, s_c)$ is the heuristic value of $Res(s_c, a_i)$, $k \in [0,1]$ is a reduction rate (usually in our tests we set $k \in [0.15, 0.5]$) to increase the transition probabilities of *Helpful Actions*.

### 3.4 Plan Evaluation

At end of each iteration a quality evaluation of all plans built by the ants is needed to perform a pheromone update. An intuitive (trivial) criterium 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 $h_{min}$, the minimum heuristic value (i.e. minimum distance from goals) reached during its execution: the smaller $h_{min}$, the higher the plan quality. Moreover, if two plans have the same $h_{min}$, then the plan which reaches first this value is better. For instance, let us consider the situation in which three ants build three different plans: *Plan A*, which reaches $h_{min} = 2$ at time step 2, *Plan B*, which reaches $h_{min} = 3$ at time step 2, and *Plan C*, which reaches $h_{min} = 2$ at time step 1. We assume that the quality of both *Plan C* and *Plan A* is better than the quality of *B* because they get lower minimum heuristic values. Moreover we assume that the quality of *C* is better than *A* because it gets the value 2 at a previous construction step.

In this way we are able to compare two different plans to decide which is better. A quantitative measure $Q(p)$ for the quality of plan $p$ can be easily defined as

$$
Q(p) = \left(\frac{1}{1 + h_{min}}\right)^\gamma \left(\frac{1}{t_{min}}\right)^\delta 
$$

where $t_{min}$ is the step in which $p$ reaches $h_{min}$, $\gamma$ and $\delta$ are parameters to tune the importance of two terms.

To consider also the number of goals reached we can add a term to the equation above:

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

where $g_{found}$ is the number of goals reached, $g_{count}$ is the total number of goals and $\theta$ is a parameter to adjust its importance.
3.5 Pheromone Updating

In our framework we 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 [11]:

\[ \tau_i^j = (1 - \rho)\tau_i^j + \rho \sum_{p \in P_{upd} | c_i^j \in p} \frac{Q(p)}{\sum_{p' \in P_{upd}} Q(p')} \]  

(5)

where \( \tau_i^j \) is the pheromone value of \( c_i^j \), \( \rho \) the pheromone evaporation rate, \( P_{upd} \) is the set of solutions (plans) involved in the update, \( Q \) is the quality plan evaluation function.

After some preliminary experiments, we decided that, for each \( p \in P_{upd} \), only the pheromone values relative to the first \( t_{min} \) actions are updated, the other ones are neglected. The main reason is that what a plan does after having reached its “best” state can be ignored because it probably moves in a wrong direction, i.e. away from the goals.

The pseudo code of the resulting algorithm, called ACOPlan, is shown in figure 1.

4 Related Works

There are several relevant planners which can be directly related to this work. First of all we have to cite LPG [4] because it is based on a stochastic algorithm. It is important to note that stochastic approaches to planning did not have the due attention by the planning community (with respect to the standard deterministic approaches) even if it has been proved they can give very good performances also in the case of optimality problems. Moreover, we have to cite heuristic planners like HSP [6] and FF [7] for two different reasons: (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 [5] because, in this case, the optimality is expressed in terms of number of planning level and the results are not comparable.

5 Experimental results

ACOPlan has been tested over some domains taken from last International Planning Competitions (IPC). In general these domains are used as standard bench-
Algorithm 1 The algorithm ACOPlan

1: \( s_{\text{best}} \leftarrow \emptyset \)
2: \( \text{InitPheromone}(T, c) \)
3: while termination condition not met do
4: \( s_{\text{iter}} \leftarrow \emptyset \)
5: for \( m \leftarrow 1 \) to number of ants do
6: \( s_p \leftarrow \emptyset \)
7: state \( \leftarrow \) initial state of the problem
8: for \( i \leftarrow 1 \) to max number of construction step do
9: \( A_i \leftarrow \) feasible actions on state
10: \( H_i \leftarrow \emptyset \)
11: \( HA_i \leftarrow \text{GetHelpfulActions}(\text{state}, A_i) \)
12: for all \( a_i^j \) in \( A_i \) do
13: \( h_{i}^{j} \leftarrow \) heuristic value of \( a_i^j \)
14: \( H_i \leftarrow H_i \cup h_{i}^{j} \)
15: end for
16: \( a_k \leftarrow \text{ChooseAnAction}(T, H_i, A_i, HA_i) \)
17: extend \( s_p \) adding \( a_k \)
18: update state
19: end for
20: if \( f(s_p) > f(s_{\text{iter}}) \) then
21: \( s_{\text{iter}} \leftarrow s_p \)
22: end if
23: end for
24: if \( f(s_{\text{iter}}) > f(s_{\text{best}}) \) then
25: \( s_{\text{best}} \leftarrow s_{\text{iter}} \)
26: end if
27: UpdatePheromone(T, s_{\text{best}}, s_{\text{iter}}, \rho) 
28: end while

marks to compare planner performances. We run a set of systematics tests over the domains Rovers, 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\(^2\).

We chose to compare ACOPlan with LPG, HSP and FF. LPG is very performant and, when run with -quality option, 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 with the 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

\(^2\) A complete repository and detailed descriptions of these domains can be found in the ICAPS website www.icaps-conference.org
a larger set of problems and collect more results.
FF has no option to choose and it runs in default version.
Also ACOplan has many parameters that have to be chosen. After some pre-
liminary tests we decided to use this setting: 10 ants, 5000 iterations, \(\alpha = 2\),
\(\beta = 5\), \(\rho = 0.15\), \(c = 1\), \(k = 0.5\), 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 names 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 guar-
tanteeing the optimality. The symbol – in table entries means that the corre-
sponding problem has not been solved in 2 hours of CPU times or because of
memory fault.
Results in Table 1 for the Driver domain show how the quality of solutions syn-
thesised by ACOplan is practically always better than the ones extracted by
FF and HSP and is often better than the ones extracted by LPG. Only in one
case LPG extracts a better solution. For instance, with respect to FF, on the
average, the percentage improvement is 15%, with a top of 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 and 10% with respect to LPG
with a top 15% in both cases.
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.

6 Conclusions and Future Works
In this paper we have described a first application of the Ant Colony Opti-
mization meta–heuristic to Optimal Propositional Planning. The preliminary
empirical tests have shown encouraging results and that this approach is a vi-
able 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 requiring 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.
<table>
<thead>
<tr>
<th>Problem</th>
<th>ACOplan</th>
<th>HSP</th>
<th>FF</th>
<th>HSP -opt</th>
<th>LPG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>length</td>
<td>time</td>
<td></td>
<td>length</td>
<td>time</td>
</tr>
<tr>
<td>Driverlog_1</td>
<td>7</td>
<td>0.01</td>
<td>7</td>
<td>0.01</td>
<td>7</td>
</tr>
<tr>
<td>Driverlog_2</td>
<td>19</td>
<td>3.87</td>
<td>24</td>
<td>0.01</td>
<td>12</td>
</tr>
<tr>
<td>Driverlog_3</td>
<td>12</td>
<td>0.06</td>
<td>14</td>
<td>0.01</td>
<td>12</td>
</tr>
<tr>
<td>Driverlog_4</td>
<td>16</td>
<td>24.32</td>
<td>21</td>
<td>0.01</td>
<td>16</td>
</tr>
<tr>
<td>Driverlog_5</td>
<td>19</td>
<td>2.79</td>
<td>22</td>
<td>0.01</td>
<td>19</td>
</tr>
<tr>
<td>Driverlog_6</td>
<td>11</td>
<td>47.35</td>
<td>13</td>
<td>0.01</td>
<td>11</td>
</tr>
<tr>
<td>Driverlog_7</td>
<td>13</td>
<td>0.4</td>
<td>15</td>
<td>0.01</td>
<td>13</td>
</tr>
<tr>
<td>Driverlog_8</td>
<td>22</td>
<td>29.50</td>
<td>26</td>
<td>0.01</td>
<td>24</td>
</tr>
<tr>
<td>Driverlog_9</td>
<td>22</td>
<td>12.87</td>
<td>28</td>
<td>0.03</td>
<td>22</td>
</tr>
<tr>
<td>Driverlog_10</td>
<td>17</td>
<td>96.56</td>
<td>24</td>
<td>0.03</td>
<td>20</td>
</tr>
<tr>
<td>Driverlog_11</td>
<td>19</td>
<td>899.56</td>
<td>24</td>
<td>0.07</td>
<td>25</td>
</tr>
<tr>
<td>Driverlog_12</td>
<td>36</td>
<td>432.68</td>
<td>44</td>
<td>0.07</td>
<td>52</td>
</tr>
<tr>
<td>Driverlog_13</td>
<td>26</td>
<td>119.61</td>
<td>33</td>
<td>0.16</td>
<td>36</td>
</tr>
<tr>
<td>Driverlog_14</td>
<td>34</td>
<td>5440.28</td>
<td>44</td>
<td>1.66</td>
<td>38</td>
</tr>
<tr>
<td>Driverlog_15</td>
<td>44</td>
<td>4920.89</td>
<td>48</td>
<td>1.65</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 1. Results for Driver domain collecting solution lengths and CPU time

<table>
<thead>
<tr>
<th>Problem</th>
<th>ACOplan</th>
<th>FF</th>
<th>LPG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rover_1</td>
<td>10</td>
<td>0.04</td>
<td>10</td>
</tr>
<tr>
<td>Rover_2</td>
<td>8</td>
<td>0.04</td>
<td>8</td>
</tr>
<tr>
<td>Rover_3</td>
<td>11</td>
<td>0.26</td>
<td>13</td>
</tr>
<tr>
<td>Rover_4</td>
<td>8</td>
<td>0.08</td>
<td>8</td>
</tr>
<tr>
<td>Rover_5</td>
<td>22</td>
<td>0.25</td>
<td>22</td>
</tr>
<tr>
<td>Rover_6</td>
<td>36</td>
<td>26.35</td>
<td>38</td>
</tr>
<tr>
<td>Rover_7</td>
<td>18</td>
<td>0.34</td>
<td>18</td>
</tr>
<tr>
<td>Rover_8</td>
<td>26</td>
<td>0.98</td>
<td>28</td>
</tr>
<tr>
<td>Rover_9</td>
<td>31</td>
<td>160.66</td>
<td>33</td>
</tr>
<tr>
<td>Rover_10</td>
<td>35</td>
<td>38.28</td>
<td>37</td>
</tr>
<tr>
<td>Rover_11</td>
<td>32</td>
<td>207.72</td>
<td>37</td>
</tr>
<tr>
<td>Rover_12</td>
<td>19</td>
<td>1.15</td>
<td>19</td>
</tr>
<tr>
<td>Rover_13</td>
<td>43</td>
<td>156.52</td>
<td>46</td>
</tr>
<tr>
<td>Rover_14</td>
<td>28</td>
<td>80.61</td>
<td>28</td>
</tr>
<tr>
<td>Rover_15</td>
<td>41</td>
<td>979.70</td>
<td>42</td>
</tr>
<tr>
<td>Rover_16</td>
<td>41</td>
<td>250.89</td>
<td>46</td>
</tr>
<tr>
<td>Rover_17</td>
<td>47</td>
<td>5126.78</td>
<td>49</td>
</tr>
<tr>
<td>Rover_18</td>
<td>41</td>
<td>4758.65</td>
<td>42</td>
</tr>
<tr>
<td>Rover_19</td>
<td>65</td>
<td>3427.45</td>
<td>74</td>
</tr>
<tr>
<td>Rover_20</td>
<td>–</td>
<td>–</td>
<td>96</td>
</tr>
</tbody>
</table>

Table 2. Results for Rovers domain collecting solution lengths and CPU time
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. Also the extension to handle preferences seems to be straightforward, being necessary only a modification in the computation of $Q(p)$.

Acknowledgements

Many thanks to IRIDIA (http://code.ulb.ac.be/iridia.home.php) Institute at Université Libre de Bruxelles for letting us use their machines to run our experiments. In particular a special thank to Dr. Thomas Stützle for his useful suggestions.

References