International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

An Experimental Consideration of the Hybrid Architecture Based on the Situated Action Generator

Serin Lee, Takashi Kubota, and Ichiro Nakatani

***Abstract*—**The approaches to make an agent generate intelligent actions in the AI field might be roughly categorized into two ways—the classical planning and situated action system. It is well known that each system have its own strength and weakness. However, each system also has its own application field. In particular, most of situated action systems do not directly deal with the logical problem. This paper first briefly mentions the novel action generator to situatedly extract a set of actions, which is likely to help to achieve the goal at the current situation in the relaxed logical space. After performing the action set, the agent should recognize the situation for deciding the next likely action set. However, since the extracted action is an approximation of the action which helps to achieve the goal, the agent could be caught into the deadlock of the problem. This paper proposes the newly developed hybrid architecture to solve the problem, which combines the novel situated action generator with the conventional planner. The empirical result in some planning domains shows that the quality of the resultant path to the goal is mostly acceptable as well as deriving the fast response time, and suggests the correlation between the structure of problems and the organization of each system which generates the action.

***Keywords*—**Situated reasoning, situated action, planning, hybrid architecture

I. Introduction

THE approaches to make an agent generate intelligent actions in the AI field might be very roughly categorized into two ways—the classical planning and situated action system. The former which comes from logical traditions is to extract the path to the given goal by reasoning about the agent's own action and situation. In particular, the planners based on (heuristic) search such as Graphplan, HSP, FF, LPG, and YAHSP have been outperformed in most of classical planning benchmarks [1]-[5].

However, the latter study on how agents use their circumstances to achieve intelligent actions, rather than reasoning actions away from its circumstances, actively arose in the 1980s as a reaction against the above classical view [7], [9]. Numerous scientific applications for adapting this notion have been followed, and usually showed the faster response time in deriving actions for the goal. As a result, this fast runtime makes the situated system be well situated compared with the classical planning in the dynamic and (partially) unknown environment [6], [8]. However, most of its practical applications have been concentrated in the navigation field of mobile robots, and not directly handled the logical problem which has been dealt by the classical plan.

To solve problems of each approach, the hybrid architecture which simply combines two methods has been proposed. However, most of them still have inherent problems of situated action approaches, and do not provide the fine granularity [6], [12].

The objective of this paper is to develop the algorithm and architecture which can situatedly solve the logical problem, which might be necessary for the agent to accomplish the complicated task in the dynamic and hazardous environment.

And thus, the situated action generator not to provide the full path to the goal, but to situatedly(immediately) derive only the proper action, which is necessary to achieve the goal of the given logical problem, could be first considered. However, the conventional planning is required to exactly extract only the action that needs to achieve goal from executable actions at the current situation. Therefore, we mention the novel situated action generator which situatedely derives the approximation of the above action from the relaxed logical space, which is similar to the relaxed plan space used in most of heuristic planners [2]-[5]. Because the relaxed logical space for deriving only the action set could be relatively small, and the extraction of the action would be also very fast, the response time of the agent to the environment is consequently fast. It is believed that this can make the agent be well situated.

However, since the above derived action is just an approximation of the action which requires to achieve the goal, if the problem includes the deadlock, then the agent only with the situated action generator could be caught in the deadlock. To avoid this situation, the conventional planning system is required.

This paper solves this problem through the newly developed hybrid architecture, which combines the novel situated action generator with the conventional planner. In this architecture, the classical planner is called on to guide the agent to the goal only when the agent might meet the deadlock.

It might be critical to decide when the conventional planner produces the proper action instead of the situated action

Serin Lee is with the University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan. (phone: +81-42-759-8311; e-mail: srlee@nnl.isas.jaxa.jp).

Takashi Kubota is with Japan Aerospace Exploration Agency-Institute of Space and Astronautical Science, 3-1-1 Yoshinodai, Sagamihara-shi, Kanagawa, Japan. (e-mail: kubota@nnl.isas.jaxa.jp).

Ichiro Nakatani is with the Department of Electronic Engineering, the University of Tokyo, and Japan Aerospace Exploration Agency-Institute of Space and Astronautical Science. (e-mail: nakatani@nnl.isas.jaxa.jp).

generator. This paper roughly regards the case in which no action is extracted by the situated action generator as the necessary criterion which the conventional planner should provide the action to the agent.

By the situated action generator or conventional planner, the action, which is necessary to achieve the goal, is exactly or approximately extracted from the executable actions at every situation. After performing the extracted action (set), the agent should recognize the situation for deciding the next likely action set. Through repeating this strategy, the agent is expected to arrive at the goal state. The consequent path to the goal could be considered as Suchman's post-hoc represented plan [7].

This paper is organized as follows. After defining notations used in this paper, we briefly present how the situated action can be generated. And then, this paper explains the novel hybrid strategy based on the situated action generator to avoid the deadlock. This paper finally provides an empirical result before conclusion.

II. Definitions

A state S is a finite set of logical atoms (facts). A planning task $P = \{O, I, G\}$ is a triple where O is the set of actions, and I (the initial state), and G (the goals) are set of atoms. An action o is STRIPS action. $\{pre(o), add(o), del(o)\}$ denotes the precondition, addition effect, and deletion effect of o. The result of applying o to a state S becomes the state $\text{Result}(S, < o >) = (S \bigcup add(o)) \setminus del(o)$ if $pre(o) \subseteq S$. Otherwise, it is undefined.

A relaxed logical space $\Re$ is built by the following assumptions.

(1) The deletion effect of actions is ignored. That is, the fact is not deleted by the action.

(2) The fact which can be achieved at some time is regarded to be achieved at the time. That is, the case in which the time to achieve the fact is delayed is not considered.

The set F_g is a set of currently achievable facts which is necessary to accomplish G. However, the conventional planner is required to exactly derive the F_g. Therefore, the approximation of F_g is derived from $\Re$, and the set F_{ag} denotes it. The set A_{ag} is a set of currently executable actions that is required to achieve each fact of F_{ag}. However, because all actions included in A_{ag} are not always applicable at the same time, a set $L \subseteq A_{ag}$ which has a precedence over other subsets of A_{ag} should be derived. That is, the agent is actually expected to arrive at the goal through applying L at every situation.

III. Conventional Planning - Graphplan

Since 1960's, various algorithms have been proposed to make an agent build the plan for itself. In particular, the search-based planner has been one of the most interesting approaches. The well-known graphplan is also one of those state space search-based planners, and has been dealt from various aspects [1].

The planning space of the graphplan is a directed graph, and includes two types of nodes, proposition nodes and action nodes, arranged into levels. Nodes in odd numbered levels correspond to action instances; there is one such node for each action instance whose preconditions(facts) are present, and are mutually consistent at the previous even numbered level [13]. All of the facts are transferred to the next level by the maintain action (*noop*). Since the specific level denotes a relative time, *mutex* (mutually exclusion relation) represents the exclusive relation in the same level.

After the planning graph is built until the last fact level includes all of goal facts that do not include mutex, the graphplan performs a backward-chaining search on the space to look for the plan.

In this paper, A_i and F_i denotes the *i*-th action and fact level, respectively. For example, A_0 denotes the set of action that is currently executable.

Figure 1 shows an example of the planning graph for the rocket problem described in PDDL.

TABLE I
Rocket Domain and Its Problem Described in PDDL

```
(define (domain rocket)
(:requirements :strips)
(:types rocket place cargo)
(:predicates (at_r ?r - rocket ?p - place) (fuel ?r - rocket)
             (in ?c - cargo ?r - rocket) (at ?c - cargo ?p - place))
(:action move
    :parameters (?r - rocket ?from - place ?to - place)
    :precondition (and (at_r ?r ?from) (fuel ?r))
    :effect (and (at_r ?r ?to) (not (at_r ?r ?from)) (not (fuel ?r))))
(:action unload
    :parameters (?r - rocket ?p - place ?c - cargo)
    :precondition (and (at_r ?r ?p) (in ?c ?r))
    :effect (and (not (in ?c ?r)) (at ?c ?p)))
(:action load
    :parameters (?r - rocket ?p - place ?c - cargo)
    :precondition (and (at_r ?r ?p) (at ?c ?p))
    :effect (and (not (at ?c ?p)) (in ?c ?r))))

(define (problem rckt_1r2p2c)
(:domain rocket)
(:objects   A – cargo
            B – cargo
            L – place
            P – place
            R - rocket)
(:init (at A L) (at B L) (at  r R L) (fuel R))
```

Recently, the planners based on heuristic search such as HSP, FF, LPG, and YAHSP have been outperformed in most of classical planning benchmarks [2]-[5].

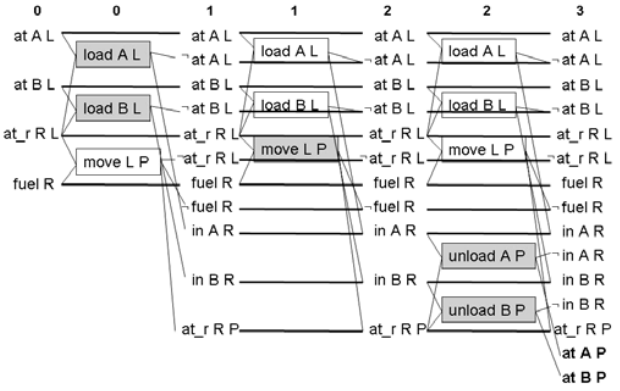

Fig. 1 Example of the planning graph

IV. Situated Action Generator

In this paper, the planning space such as described in Figure 1 is approximately built to immediately derive not the entire course to the goal, but the situated action on every situation.

A. Relaxed Logical Space

The relaxed logical space $\Re$ is similar to the relaxed planning graph space used in some heuristic planners such as FF [3]. As we defined in the previous section, since it is assumed in $\Re$ that the fact is not deleted by the action, the deletion effects of actions are ignored. Also, since the case in which the time to achieve the fact is delayed is also ignored in building $\Re$, if the action o is included in the $level_i$ of the planning graph space, then o is not appeared in the $level_j$ $(i < j)$.

As a result, $\Re$ can be considered as the graphplan space built by

- $P' = (O', S, G)$,

where $O' = \{pre(o), add(o), \varnothing \mid (pre(o), add(o), del(o)) \subset O\}$, and S denotes the current state.

- If $o \in level_i$, then o is not appeared in the $level_j$ $(i < j)$.
- The last fact level includes all of goal facts.

An example on the relaxed logical space for the rocket problem is shown in Figure. 2.

B. Extraction of A_{ag} and L

After building the relaxed logical space $\Re$, the simple backward chaining is performed from each goal fact to the action level 0. Since this space does not include the deletion (negation) fact, there is no exclusive relation (*mutex*) between two actions or facts.

Note that $\Re$ does not contain the fact which the time to achieve it is delayed. Therefore, during this backward chaining, if there is a noop for achieving a fact, then the noop is first selected [3]. Otherwise, a non-noop is randomly selected because the actual shortest path to the goal over the entire level cannot be immediately derived from $\Re$.

In this paper, we regard the element extracted from F_1 and A_0 by the above backtracking strategy through *noop first heuristic with randomly selected non-noops* as F_{ag} and A_{ag}, respectively. This A_{ag} has some similarities with heuristic actions proposed in various planners such as the helpful action of FF [3].

After deriving A_{ag} from the relaxed logical space $\Re$, L which has a precedence over other subsets of A_{ag} is directly extracted from A_{ag} by the following algorithm.

(1) The candidate set of L, L_{cand} is initialized to A_{ag}.

(2) If some action in L_{cand} deletes one of the precondition of other actions, then the action is removed from L_{cand}. For example, if $o_i \prec o_j$, then o_j cannot become the element of L.

(3) If some action in L_{cand} deletes the element of F_{ag} which is added by the action o in L_{cand}, o is removed from L_{cand}.

(4) The noop in L_{cand} is removed.

(5) The candidate set L_{cand} becomes L.

In Figure 2, $A_{ag} = A_0 = \{$(load A L), (load B L), (move L P)$\}$, and $L = \{$(load A L), (load B L)$\}$.

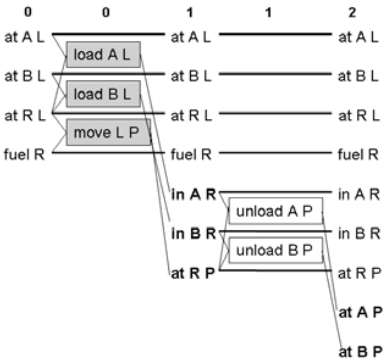

Fig. 2 Example of the relaxed logical space $\Re$

The entire description of the situated action generator is described in Figure 3.

```
while 𝒢 ⊈ ℱ_i of ℛ do
  building the next level of ℛ with i ← i + 1
endwhile
let 𝒢_s1 = ∅
for level = i, ···, 1 do
  if level = i then
     𝒢_s2 = 𝒢
  endif
  else
     𝒢_s2 = 𝒢_s1
  let 𝒜_ag = ∅
  forall f ∈ 𝒢_s2 do
     let 𝒜_ag−cand = ∅
     forall o ∈ 𝒜_level−1 do
       if f ∈ add(o) ∧ o = noop then
          𝒜_ag ← 𝒜_ag ⊕ o
          𝒢_s1 ← 𝒢_s1 ⊕ pre(o)
       endif
       if f ∈ add(o) ∧ o = non noop ∧ 𝒜_ag = ∅ then
          𝒜_ag−cand ← 𝒜_ag−cand ⊕ o
       endif
     endfor
     if 𝒜_ag = ∅ then
       𝒜_ag ← 𝒜_ag⊕ randomly selected o from 𝒜_ag−cand
       𝒢_s1 ← 𝒢_s1 ⊕ pre(o)
     endif
  endfor
endfor
let ℒ = ∅, ℒ_cand = 𝒜_ag
forall o_i, o_j ∈ ℒ_cand do
  if o_i ≺ o_j then
     ℒ_cand ← ℒ_cand ⊖ o_j
  endif
  if ∀f ∈ add(o_i) : ∃f : f ∈ 𝒢_s2 ∧ f ∈ del(o_j) then
     ℒ_cand ← ℒ_cand ⊖ o_i
  endif
endfor
ℒ ← ℒ ⊕ ℒ_cand
return ℒ
```

Fig. 3 Situated Action Generator

In building $\Re$, the case in which the time to achieve the fact of F_{ag} is delayed is not considered. However, there is some case which some facts of F_{ag} must be delayed.

If the decision on which facts should be delayed is not made, then the agent with the situated action generator could be caught in the deadlock or circular routine.

To escape the circular routine in the case of deadlock free, the randomness should be added to the situated action generator. Therefore, the way in which one action is randomly selected from A_{ag} with the probability ζ or from $A_0 - A_{ag}$ with $1-\zeta$ when $L = \varnothing$ can be one of the proper solutions.

V. Hybrid Architecture

In the case of deadlock free problems, the randomness is added to deal with the case in which the time to achieve the fact of F_{ag} should be delayed. However, in the case of the problem including deadlocks, if the above case is not more carefully dealt, then the agent could catch into the deadlock.

Therefore, we should first modify the algorithm deriving L to remove all of the potentially risky actions from the selection of L. This modification can be realized by adding the routine to exclude the action which deletes the precondition of other actions in A_0, and this is a necessary criterion to be caught in the deadlock by performing the action. In the remaining part of this paper, L is assumed to be extracted by this method.

As we have mentioned in the previous section, the conventional planner could help the situated action generator avoid falling into the deadlock. Therefore, the hybrid architecture which combines two approaches could be considered to realize it. However, it might be critical to decide when the conventional planner produces the proper action instead of the situated action generator. This paper deals with this problem through the following novel *three-layered hybrid architecture*.

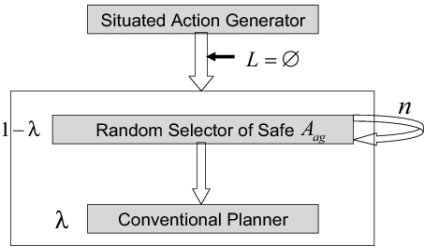

Fig. 4 Proposed Hybrid Architecture

If no element is finally included in L from the situated action generator, then the second or third layer provides the proper action to the agent.

At this time, the second layer is selected to build the plan $P' = (O, I', G')$ with the probability $1-\lambda$, where S is the current state, o_r is a randomly selected action from the set A_{ag}, $I' = (S, < o_r >) = (S \bigcup add(o_r)) \setminus del(o_r)$ is the initial state, $\Pr = \{pre(o) \mid o \in A_0\}$ is the set of the precondition of currently executable actions, and $G' = (\Pr - pre(o_r)) \bigcup del(o_r)$.

That is, the second layer is to randomly select the safe action that does not permanently prohibit the execution of currently executable other actions.

Of course, although the problem does not include deadlocks, there could not exist P'. However, the existence of P' for some action o_r means that the agent is not caught into the deadlock by executing o_r. Therefore, if the above plan of P' exists, then the o_r can be selected to be executed by the agent. Otherwise, the above plan is rebuilt for other actions. This paper assumes that this re-plan for other actions is performed with *n* times.

In our experience, although the action randomly derived from P' is very rough to achieve the goal, it can be obtained fast since most of P' is much smaller than original planning task P. Therefore, P' could derive better situatedness than the case in which only P is considered.

If P' cannot be discovered even after *n* trials, then the

third layer, conventional planner provides the action to the agent. Although the conventional planner builds the entire course to the goal, this paper assumes that only the first action of the course is selected for the agent.[1]

VI. Experimental Results

We compared the proposed hybrid architecture with various parameters (λ and n), and a well-known classical STRIPS planner, FF v2.3. The original FF planner was embedded in the hybrid architecture as a conventional planner, and also modified for re-planning experimentations. They are all implemented in C, and no other particular library is used.

In this paper, we first compared the average of response times of the hybrid architecture and modified FF planner for re-planning, which takes to derive the action set (or an action) at every situation until achieving the goal. And then, the qualities, that is, lengths of the *post-hoc represented* plan were compared.

We used two domains in our experimentations: the Logistics domain, and the Freecell domain. The Logistics domain does not include the deadlock, but the Freecell domain includes it.

The maximum amount of time which is allowed for each problem was fixed to 1600 seconds. We set (n, λ) to (3, 0.8), (3, 0.5), (3, 0.2), and (9, 0.5).

The experimentation was carried on a Pentium IV 1.7GHz machine with 768M of RAM running Linux 2.6. The results are described in the following figures.

As we briefly stated in the previous section, the experimental result shows that the modified FF re-planner can be caught in the circular routine. All cases on the below results in which the FF re-planner could not solve the problem were caused by being caught in the circular routine.

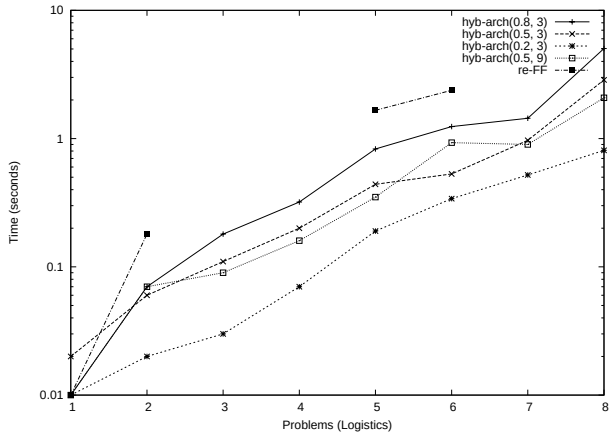

Fig. 5 Time on Logistics problems

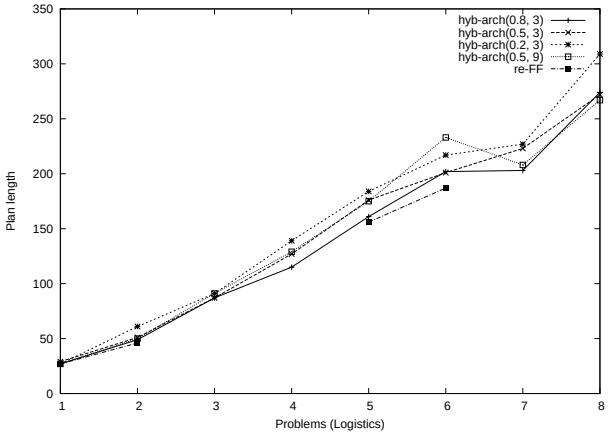

Fig. 6 Plan length on Logistics problems

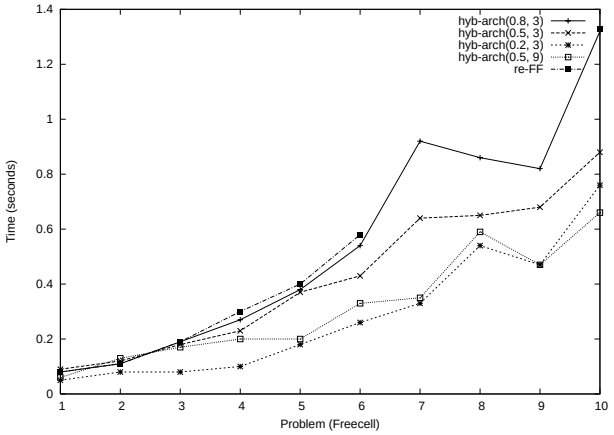

Fig. 7 Time on Freecell problems

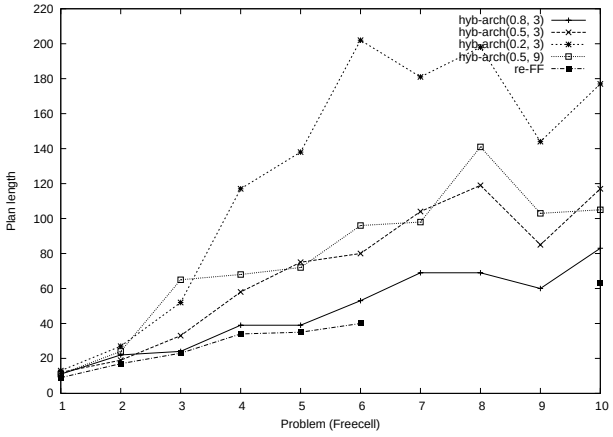

Fig. 8 Plan length on Freecell problems

As it can be easily guessed, the more the conventional planner takes charge of providing the action, the better planning quality can be obtained. But, it would also require much time, and thus could not provide the better situatedness.

[1] Some of heuristic planners such as FF can be caught in the circular routine if they are used in re-planning by this method.

This tendency is more apparent in the case of changing λ, than *n*.

VII. CONCLUSION

We first briefly presented the novel action generator to situatedly extract a set of actions, which is likely to help to achieve the goal at the current situation. To derive it, we made two assumptions : the fact is not deleted by the action, and the fact which can be achieved at some time is regarded to be achieved at the time. Under these assumptions, the approximation of the set of currently achievable facts which is necessary to accomplish the goal is derived, and then the action to properly achieve the fact set is extracted.

However, without the help of the conventional planner, the situated action generator could be caught in the deadlock. Therefore, the hybrid architecture which combines two approaches presented to realize it.

The empirical result showed that the quality of the post-hoc reconstructed plans is relatively acceptable as well as deriving the fast response to the situation. In particular, the performance on the deadlock free domain was much better than that of the domain including the deadlock.

It might be desired that the rate with which the conventional planner is selected in extracting the action is changed with the domain. However, it is well known that the decision of the deadlock free is PSPACE [3], [14]. Therefore, this paper regarded the case in which the approximation of the currently executable action required to achieve the goal can be discovered by the situated action generator as the necessary criterion of the deadlock free. To enhance the response time, more elaborate necessary condition should be discovered.

REFERENCES

[1] Avrim L. Blum and Merrick L. Furst "Fast Planning Through Planning Graph Analysis" , *Artificial Intelligence*, vol. 90, 1997
[2] B. Bonet and H. Geffner, "Planning as Heuristic Search" *Artificial Intelligence*, vol. 129, 2001
[3] Jörg Hoffman and Bernard Nebel, " The FF Planning Systems: Fast Plan Generation Through Heuristic Search", *Journal of Artificial Intelligence Research*, vol.14, 2001
[4] Vincent Vidal, "A Lookahead Strategy for Heuristic Search Planning", *Proc. AAAI-04*, 2004
[5] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina, "Planning through Stochastic Local Search and Temporal Action Graphs in LPG", *Journal of Artificial Intelligence Research*, vol. 20, 2003
[6] Ronald C. Arkin, Behavior-Based Robotics, The MIT Press, 1998
[7] Lucy Suchman, Plans and Situated Actions - The Problem of Human-Machine Communication, Cambridge University Press, 1987
[8] Rodney A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, vol. 2, 1986
[9] Robert A. Wilson and Frank C. Keil, The MIT Encyclopedia of the Cognitive Sciences, The MIT Press, 1999
[10] John McCarthy, "Artificial Intelligence, Logic and Formalizing Common Sense", *Philosophical Logic and Artificial Intelligence*, ed. R. Thomason, Kluwer Academic, 1989
[11] Jana Koehler and Jörg Hoffman, "On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm", *Journal of Artificial Intelligence Research*, vol. 12, 2000
[12] R. James Firby, "An investigation into reactive planning in complex domains", *Proc. the 6th National Conference on AI*, 1987
[13] Daniel S. Weld, Recent Advances in AI Planning, AI Magazine, 1999
[14] T. Bylander, "The computational complexity of propositional STRIPS planning", *Artificial Intelligence*, vol. 69, 1994