Int. J. Appl. Math. Comput. Sci., 2005, Vol. 15, No. 3, 359–367

NON–COOPERATIVE GAME APPROACH TO MULTI–ROBOT PLANNING

ADAM GAŁUSZKA, ANDRZEJ ŚWIERNIAK

Silesian University of Technology
Institute of Automatic Control
Akademicka 16, 44–100 Gliwice, Poland
e-mail: {agaluszka, aswierniak}@ia.polsl.gliwice.pl

A multi-robot environment with a STRIPS representation is considered. Under some assumptions such problems can be modelled as a STRIPS language (for instance, a Block World environment) with one initial state and a disjunction of goal states. If the STRIPS planning problem is invertible, then it is possible to apply the machinery for planning in the presence of incomplete information to solve the inverted problem and then to find a solution to the original problem. In the paper a planning algorithm that solves the problem described above is proposed and its computational complexity is analyzed. To make the plan precise, non-cooperative strategies are used.

Keywords: planning problems, multi-robot environment, STRIPS language, non-cooperative games, planning complexity

1. Introduction

In multi-agent (multi-robot) environments each agent tries to achieve its own goal. This leads to complications in problem modelling and searching for a solution: in most cases, agents' goals are conflicting: the agents usually have different capabilities and goal preferences, and they simultaneously interact with the problem environment. Many methods for solving these problems are known and developed in the literature (Belker *et al.*, 2002; Boutilier and Brafman, 2001; Kraus *et al.*, 1998). Most approaches take into account situations where agents in conflicting situations cooperate to make decisions (Karacapilidis and Papadias, 1998; Kraus *et al.*, 1998; Zhang *et al.*, 2001). In this paper the opposite situation is considered, i.e., agents cannot cooperate.

In our case the problem environment was modelled as the Block World with a STRIPS representation. This domain is often used to model planning problems (Boutilier and Brafman, 2001; Gałuszka and Świerniak, 2003; 2003a; Kraus *et al.*, 1998; Nilson, 1980; Slaney and Thiebaux, 2001; Smith and Weld, 1998) because of complex operator interactions and simple physical interpretations.

Since the 1970s, the STRIPS formalism introduced by (Nilson, 1980) has become popular for planning problems (Weld, 1999). Planning problems are PSPACE-complete in a general case (Baral *et al.*, 2000; Bylander, 1994), and even in a Block World environment they are not easy (here the problem of optimal planning is NP-complete, cf. Gupta and Nau, 1993)). This Block World is stated now as an experimentation benchmark for planning algorithms (Howe and Dahlman, 2002).

1.1. Contribution of the Paper

In this paper we propose a methodology that solves the problem of plan generation for robots being in conflict. This methodology combines the STRIPS language and game theory: it is based on a block world environment, the invertibility of STRIPS planning problems, conformant planning and non-cooperative games. The STRIPS domain is modified according to a classical one in such a way that generating a conformant plan is easy in the sense of computational complexity. The modification also guarantees that the plan exists but makes it imprecise. To specify the plan, the use of the Nash equilibrium is proposed.

1.2. Motivation

The Block World environment is chosen to illustrate the proposed approach despite the fact that it has been considered in the literature for decades (Nilson, 1980). Today this domain can be a representation for logistic problems, where moving blocks correspond to moving different objects like packages, trucks and planes (Slaney and Thiebaux, 2001). The case of the Block World problem where the table has a limited capacity corresponds to a container-loading problem (Slavin,

1996). In real situations decision problems at container terminals are more complex and divided into several groups: the arrival of a ship, the unloading and loading of the ship, the transport of containers from and onto the ship, the stacking of containers (see, e.g., `www.ikj.nl/container/decisions.html`).

Since the arrival of a ship and the transport of containers are usually treated as scheduling and allocation problems (Bish *et al.*, 2001; Imai *et al.*, 2001), problems of loading and unloading and container stacking can be treated as planning ones (e.g. Avriel *et al.*, 1998; Wilson 2000). In a natural way, containers can be treated as blocks and cranes as robots that are stacking and unstacking the blocks. There are also other sources of conflicting situations:

- the unloading time should be minimized (this corresponds to the minimization of the robots' moves), but a good distribution of containers on the ship should be respected because of its stability (Avriel *et al.*, 2000);
- different robots have different goals (cranes can be designed to realize different export and import container tasks in parallel).

We propose to treat these conflicts as a discrete non-cooperative game. This assumption can be justified as follows:

- firstly, as a game with nature since the ship stability is 'non-negotiable',
- secondly, the number of imported containers is known in practice shortly before the arrival of the ship (Wilson, 2000).

In these games, robot decisions are moves of blocks and the profit describes the preference of achieving a single subgoal.

In a general case (in other logistic problems), this assumption can also be justified in the case where in the robot environment communication is not allowed or the communication equipment is broken down.

1.3. Organization of the Paper

The paper is organized as follows: In Section 2 the problem is defined. In Section 3 a method of finding a solution of the defined problem is described. This section is divided into four subsections that describe the steps and an exemplary simulation. The computational complexity of the search for the solution is analysed in Section 4. Then the work is concluded.

2. Problem Definition

We focus on the following situation:

- in the initial state there are a finite number of blocks and a table with unlimited memory;
- two (or, in a general case, more) robots tend to rebuild the initial state, each on its own (each robot wants to achieve its own desired goal situation);
- the goal of each robot consists of subgoals;
- each subgoal has its preference (the preference describes the profit of achieving this subgoal);
- robots may have different capabilities (i.e., not all robots may be able to move all blocks);
- robots cannot cooperate.

We are interested in the following two problems:

- to find a solution for the above situation;
- to analyse the computational complexity of the proposed algorithm of searching for this solution.

The solution is understood as an action sequence that achieves some or all of subgoals of each robot in such a way that the sum of the profits of the robots is maximised. Because of their autonomy, the robots are also called agents.

3. Method of Finding a Solution

The problem where there are some possible initial states and one goal state is called the problem of planning in the presence of incompleteness (Weld *et al.*, 1998). The inverse problem is connected with the situation with one initial state and more possible goal states. It corresponds to a multi-robot Block World problem where each robot wants to achieve its own goal. If we are able to find a plan for the problem of planning in the presence of incompleteness, then it is possible to extract a solution for the multi-agent problem. Below we define the STRIPS language, an invertible planning problem and inverse operators.

3.1. STRIPS Language

In general, the STRIPS language is represented by four lists $(C; O; I; G)$ (Bylander, 1994; Nilson, 1980):

- a finite set of ground atomic formulae (C), called *conditions*,
- a finite set of operators (O),
- a finite set of predicates that form the initial state (I),
- a finite set of predicates that define the goal state (G).

The initial state describes the physical configuration of the blocks. The description should be complete, i.e., it should deal with every true predicate corresponding to the state. The goal state is a conjunction of predicates. Predicates in I and G are ground and function free. In a multi-agent environment each agent defines its own goal. This description does not need to be complete. The algorithm results in an ordered set of operators (i.e., an action sequence) that transforms an initial state into a goal situation.

Operators O in the STRIPS representation consist of three sublists: a precondition list (pre), an add list (add) and a delete list (del). Formally an operator $o \in O$ takes the form $pre(o) \rightarrow add(o), del(o)$. The precondition list is a set of predicates that must be satisfied in the world state to perform this operator. The delete list is a set of predicates that become false after executing the operator, and the add list is a set of predicates that become true. The last two lists represent the effects of the operator executing in the problem state. Following (Koehler and Hoffmann, 2000), the set of actions in a plan is denoted by P^o.

It is assumed that agents may have different capabilities (i.e., they can deal with limited problem elements), but this is not obligatory. No negotiations are allowed. Goal preferences are also considered. We will understand the profit as the sum of the preferences of subgoals being satisfied.

To analyze our problem, we introduced four operators similar to classical ones in the Block World as in (Nilson, 1980). The *only* difference is that the operators *stack* and *unstack* specify only the block that is currently being transformed (i.e., they do not specify on which block a transformed block is stacked, nor from which block it is unstacked):

- *pickup*(x) — the block x is picked up from the table; *precondition list & delete list: ontable*(x)*, clear*(x)*, handempty add list: holding*(x)
- *putdown*(x) — the block x is put down on the table; *precondition list & delete list: holding*(x) *add list: ontable*(x)*, clear*(x)*, handempty*
- *stack*(x) — the block x is stacked on any other block; *precondition list & delete list: holding*(x)*, clear*$(\_)$ *add list: handempty, on* $(x, \_)$*, clear*(x)
- *unstack*(x) — the block x is unstacked from any other block; *precondition list & delete list: handempty, clear*(x)*, on*$(x, \_)$ *add list: holding*(x)*, clear*$(\_)$.

The underscore sign denotes 'any block'.

Now an invertible planning problem is introduced (Koehler and Hoffmann, 2000). The problem $(C,\ O,\ I,\ G)$ is called *invertible* if and only if

$$\forall s : \forall P^O : \exists \overline{P}^O : Result(Result(s, P^O), \overline{P}^O) = s,$$

where

$Result(S, \langle\rangle) = S$,
$Result(S, \langle o\rangle) = (S(add(o))\ del(o)$ if $pre(o) \subseteq S$
S in the opposite case,
$Result(S, \langle o_1, o_2, \ldots, o_n\rangle)$
$= Result(Result(S, \langle o_1\rangle), \langle o_2, \ldots, o_n\rangle)$,

and $\overline{P}^O$ is called an *inverted plan*.

Now an inverse operator is introduced following (Koehler and Hoffmann, 2000). An operator $\overline{o} \in O$ is called *inverse* if and only if it has the form $pre(\overline{o}) \rightarrow add(\overline{o}), del(\overline{o})$ and satisfies the conditions

1. $pre(\overline{o}) \subseteq pre(o) \cup add(o) \backslash del(o)$,

2. $add(\overline{o}) = del(o)$,

3. $del(\overline{o}) = add(o)$.

Under the closed-world assumption, applying an inverse operator leads back to the previous state. Koehler and Hoffmann (2000) proved that if for each operator there exist an inverse operator, then the problem is invertible.

It is easy to see that *unstack* is an inverse operator for *stack*, and *pickup* is an inverse operator for *putdown*. We have defined the Block World as an invertible planning problem because it allows us to apply planning in the presence of incompleteness as a methodology to search for a solution to the inverted multi-agent problem and then to extract a solution for the right multi-agent problem.

3.2. Plan in the Presence of Incompleteness as an Inverted Plan in a Multi-Robot Environment

There are many possible types of modelling uncertainty in planning problems. One of the groups of planning algorithms in the presence of incompleteness deals with planning problems with uncertainty in the initial state (Weld *et al.*, 1998). In this case the algorithm seeks to generate a robust plan by thinking over all possibilities. This approach is called *conformant planning* (Smith and Weld, 1998). Conformant planning algorithms develop non-conditional plans that do not rely on sensory information, but still succeed no matter which of the allowed states the world is actually in.

3.3. Examples

The Block World environment was implemented using the PDDL language (Planning Domain Definition Language),

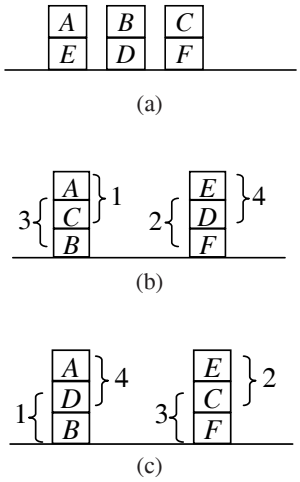

Fig. 1. Definition of Problem 1 (numbers describe the preference of achieving a subgoal) : (a) initial state, (b) desired goal state of Robot 1 (the goal conflicts with the goal of Robot 2), (c) desired goal state of Robot 2 (the goal conflicts with the goal of Robot 1) .

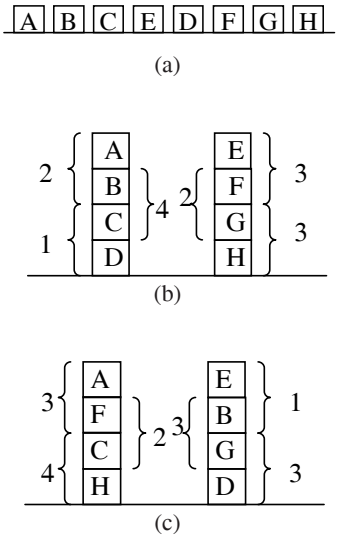

Fig. 2. Definition of Problem 2 (numbers describe the preference of achieving a subgoal) : (a) initial state, (b) desired goal state of Robot 1 (the goal conflicts with the goal of Robot 2), (c) desired goal state of Robot 2 (the goal conflicts with the goal of Robot 1).

extended for handling uncertainty in the initial state (Yale Center, 1998). The Graphplan algorithm was used to solve block world problems with uncertainty in the initial state (`www.cs.washington.edu/research/projects/www/sgp.html`).

Two different problems are presented below. In both cases two robots operate in the environment. In Problem 1 (see Fig. 1) Robot 1 is capable of moving the blocks A, B and C, whereas Robot 2 can move the blocks D, E and F. In Problem 2 (see Fig. 2) Robot 1 is capable of moving the blocks A, B, C and D, whereas Robot 2 can move the blocks E, F, G and H. In both cases the definitions of the operators are inverted (operator names are changed, i.e., *unstack* for *stack* and *pickup* for *putdown*). This implies that the plan for the inverted problem is extracted just by executing the found plan in reverse order. In both cases the agents' goals are in conflict. The case when the goals do not conflict with one another in a multi-agent environment was investigated in (Gałuszka and Świerniak, 2002).

The solution to Problem 1 *(a conformant plan)* consists of 7 steps and should be read in reverse order:

Step 7 – ((((STACK2 E)))

Step 6 – (((PICK-UP2 E)) ((STACK1 A)))

Step 5 – (((STACK2 D)) ((UNSTACK1 A)))

Step 4 – (((PICK-UP2 D)) ((STACK1 C)))

Step 3 – (((PUT-DOWN2 D)) ((UNSTACK1 C)))

Step 2 – (((PICK-UP2 D)) ((PUT-DOWN1 B)))

Step 1 – (((UNSTACK1 B))))

The solution to Problem 2 consists of 6 steps:

Step 6 – ((((STACK2 E)) ((STACK1 A)))

Step 5 – (((PICK-UP2 E)) ((PICK-UP1 A)))

Step 4 – (((STACK2 F)) ((STACK1 B)))

Step 3 – (((PICK-UP2 F)) ((PICK-UP1 B)))

Step 2 – (((STACK2 G)) ((STACK1 C)))

Step 1 – (((PICK-UP2 G)) ((PICK-UP1 C))))

Both agents can apply the above plan to attain their goals. However, when they are trying to achieve their goals simultaneously, they are in conflict. To make decisions in the presence of such a conflict situation, a non-cooperative equilibrium is proposed. Non-cooperative games have been used in planning earlier (see, e.g., Isil Bozma and Koditschek, 2001; Skrzypczyk, 2005), but here this idea is combined with the classical STRIPS system. Now we define the non-cooperative equilibrium (the Nash equilibrium, cf. Mesterton-Gibbons, 2001; Mc Kinsey, 1952), and indicate how the agents can maximise their profits (the sum of the preferences of satisfied goals) by achieving a non-cooperative (Nash) equilibrium.

3.4. Non-Cooperative Game

For our problem a plan exists only if the operators *stack* and *unstack* have only one parameter so they do not specify from which and on which block the transformed one is

stacked or stacked out. This implies that both robots can apply the found plan to reach theirs goals but not simultaneously. Using goal preferences it is possible to express this conflict with a bimatrix game represenattion (for our two-robot problem), and then to use a Nash equilibrium strategy to specify how to apply the plan simultaneously and maximise the profit (the sum of satisfied goals preferences). The analysis of the problem leads to two remarks:

Remark 1. It is not always possible to find a Nash strategy for the defined problems, and in a general case it depends on the problem size.

Remark 2. More precisely, the Nash strategy (if it exists) defines an equilibrium for the whole plan when the number of *the stack* operators in the found plan is *even* for each agent (two operators for each agent in Problem 1). When this condition is not satisfied (three operators for each agent in Problem 2), then the Nash strategy defines an equilibrium only for a part of the problem.

The conflict between two robots will be presented by a bimatrix game (Mesterton-Gibbons, 2001). The matrix A characterises the profits of the first agent, and the matrix B characterises the profits of the second agent. We assume that Agent 1 chooses rows and Agent 2 chooses columns of the matrices. The agents try to maximise profit functions defined by the matrices $A = a_{ij}$ and $B = b_{ij}$.

Definition of the Nash equilibrium. The strategy $\{i_0, j_0\}$ determines a non-cooperative (Nash) equilibrium in a bimatrix game (A, B) if the following inequalities are satisfied:

$$a_{i_o j_o} \geq a_{i j_o}, \quad b_{i_o j_o} \geq b_{i_o j}$$

for all $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

The Nash strategy is reasonable in conflict situations where players (in this case agents) strive to maximise profits, do not cooperate and make decisions independently.

Now we define matrices for Problem 1. The strategies in the matrices correspond to the plan that solves Problem 1. Agent 1 can stack the block C either on B or F, and the block A on C or D, whereas Agent 2 can stack block D on B or F and block E on C or D. The values in the matrices correspond to goal preferences (e.g., if Robot 1 stacks the block A on C and Robot 2 block D on F, then the profit of Robot 1 is 5 since it satisfied two of its subgoals, whereas the profit of Robot 2 is 0 as it satisfied none of its subgoals). Tables 1 and 2 show the profits of Agent 1 and 2.

In this game we found one strategy that satisfies the non-cooperative (Nash) equilibrium definition (in brackets). This strategy modifies the plan in such a way that

Table 1. Matrix A (profits of the first agent).

Robot 1 / Robot 2	stack D B	stack D F	stack E C	stack E D
Stack C B	3	$3+2$	(3)	$3+4$
Stack C F	0	2	0	4
Stack A C	1	$1+2$	1	$1+4$
Stack A D	0	2	0	4

Table 2. Matrix B (profits of the second agent).

Robot 1 / Robot 2	stack D B	stack D F	stack E C	stack E D
stack C B	1	0	(2)	0
stack C F	$1+3$	3	$2+3$	3
stack A C	1	0	2	0
stack A D	$1+4$	4	$2+4$	4

Agent 1 should place the block C on B and Agent 2 should place block E on C. This leads to the situation when the final state for Problem 1 takes the form presented in Fig. 3.

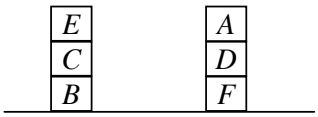

Fig. 3. Final state for Problem 2 that comes from a Nash equilibrium.

Finally, the profit of Agent 1 is now $3 + 2 = 5$, and for Agent 2 it is $2 + 4 = 6$.

Finding a solution is more difficult for real-size problems. There are two sources of this situation: a game solution is non-unique (i.e., there are Nash equilibria that cannot be compared) or it does not exist (i.e., Nash equilibria do not exist). In the first case an additional criterion could be introduced (e.g., a greedy strategy that maximises the sum of profits of all players—for greedy strategies, see, e.g., (Papadimitriou, 2001a)—or a *fair-arbiter* one based on other strategies, cf. (Skrzypczyk, 2005). In the second case a safe strategy can be applied (i.e., the strategy that maximizes the minimal possible profit (Mesterton-Gibbons, 2001; Basar and Olsder, 1982).

4. Computational Complexity of the Search for a Solution

The complexity analysis should take into account the complexity of finding a plan problem (Section 4.1) and the complexity of solving a game (Section 4.2).

4.1. Complexity of Finding a Plan

In general, planning with complete information is PSPACE-complete (Bylander, 1994). In the Block World the problem of optimal planning is NP-complete (Bylander, 1994). Planning in the presence of incompleteness belongs to the next level in the hierarchy of completeness (Baral *et al.*, 2000). The proposed algorithm is formulated in such a way that for a subclass of block world problems it reduces the complexity of finding a solution to the P class.

Non-optimal planning in the Block World is in the P class of complexity (Gupta and Nau, 1992). With no loss of generality, while analysing the complexity in our case, it is assumed that the planning problems are limited only to a completely decomposed initial state (i.e., all blocks are on the table) as it shown in Fig. 4. Then the inverted problem is to decompose all possible initial states (i.e., the goal definition consists only of 'on-table' predicates). The number of possible initials corresponds to the number of robots. So the inverted problem is planning in the presence of incompleteness (see Fig. 5).

Now there will be shown a sub-class of block world problems for which finding a plan is easy. In this class each block has the same position in the stack in each possible initial state. This class belongs to the same class of complexity as classical block world planning.

The Hass Diagram (Gupta and Nau, 1992) will be used to represent possible initial states of the block world. This diagram is a directed acyclic graph whose nodes are the blocks and arcs are from the block x to the block y if and only if on (x, y) is in an initial state. This diagram can be constructed in linear time (Gupta and Nau, 1992). Since the number of possible initial states is equal to the number of robots, the time necessary to built a Hass diagram for all initial states is also linear.

Next, for each block in each possible initial state, the position in stack is calculated using Hass diagrams. This corresponds to the problem of finding the length of a path in an acyclic graph. In Problem 2 the block positions are:

for A and E – 3,

for F and B – 2,

for C and G – 1,

for D and H – 0

for both possible initial states.

If each block has the same position in a stack in each possible initial state, then there exists the same plan for each agent that solves the problem of decomposing all initials. Hence, in order to find a plan, only the goal situation and block positions can be considered. Subgoals are serialised in decreasing order according to block positions. For the situation in Problem 2 we have the order

$$\big\{\text{(on-table A), (on-table E), (on-table F), (on-table B),}$$

$$\text{(on-table C), (on-table G), (on-table D), (on-table H)}\big\}.$$

If all blocks are on the table, then all are 'clear'. Let us redefine the order:

$$\big\{\text{(clear A), (clear E), (clear F), (clear B),}$$

$$\text{(clear C), (clear G), (clear D), (clear H)}\big\},$$

and define one operator *move-to-table* (x): *pre:*(clear x), (on $x\,y$), *add:* (clear y). The operator consists of positive preconditions and one postcondition (so non-optimal planning is tractable, cf. (Bylander, 1994)). This simple operator extends the definition of the current state taking into account the order in the goal definition, until all 'clear' predicates are true in the current state.

Each step requires only a polynomial time, so the presented planning problem is solved in polynomial time (it belongs to the complexity P class).

4.2. Complexity of Solving a Game

Now the complexity of solving a game is analysed. The complexity of the problem of finding Nash equilibria is widely described in the literature and solved only for specific cases (Papadimitriou, 2001; Fabricant *et al.*, 2004). In this case assume that k is the number of robots, n_i is the number of decisions of the i-th robot, $i = 1, 2, \ldots, k$, and $n_{max} = \max\{n_1, n_2, \ldots, n_k\}$.

To find all possible Nash equilibria for $k = 2$, the number of maximizations of $n_{\max}$ elements is $n_{\max}^2$ in the worst case. S_o, the time needed for this, is bounded by $O(n_{\max}^3)$. In general, there are $n_{\max}^k$ maximizations of $n_{\max}^{(k-1)}$ elements, so for a bounded number of robots the problem of finding Nash equilibria in this case is polynomial time complete.

5. Conclusion

Defining a Block World environment as an invertible STRIPS planning problem allows us to apply planning in the presence of incompleteness as a machinery of searching for a solution of the inverted multi-agent problem and then the extraction of a solution for the primary multi-robot problem. It is possible to use a non-cooperative

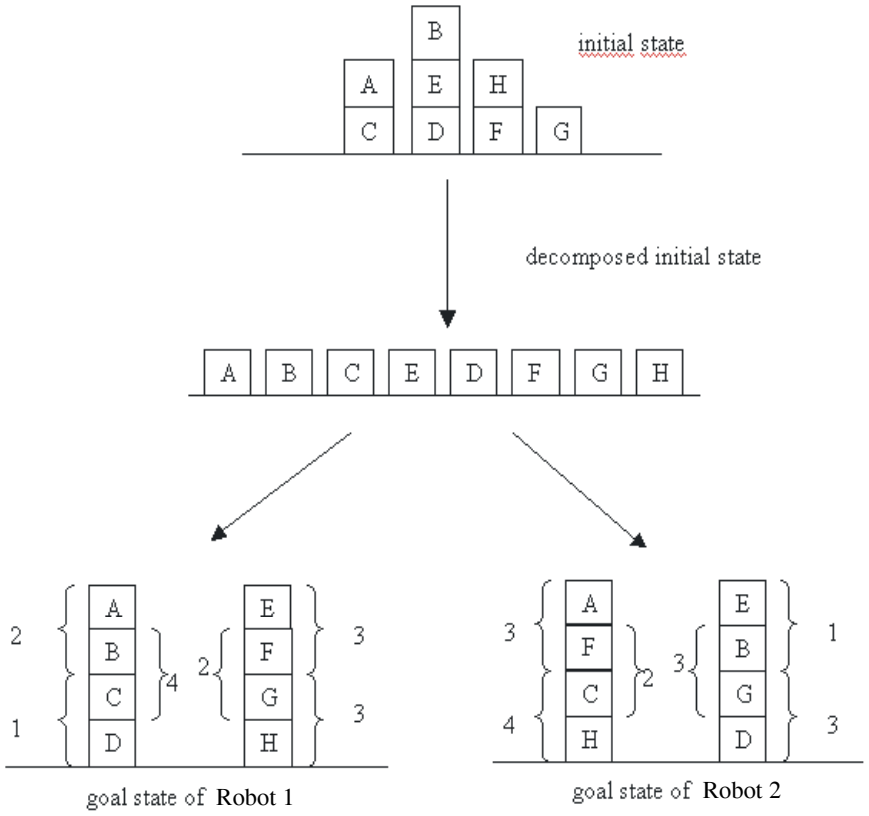

Fig. 4. Problem with a decomposed initial state.

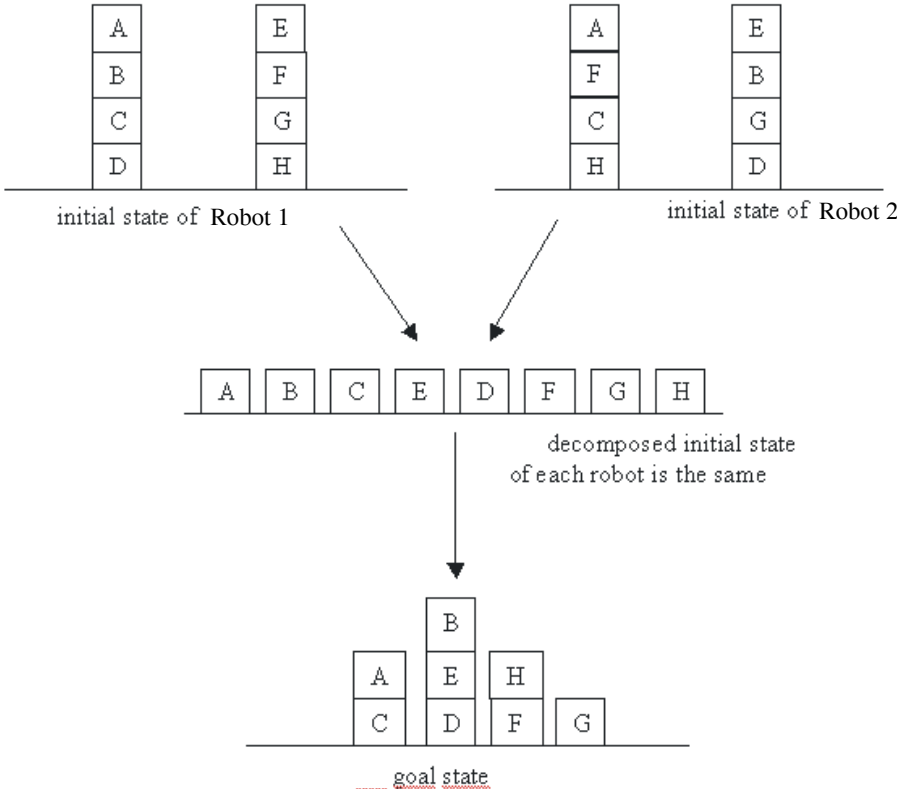

Fig. 5. Inverted problem with a disjunction of the initial state.

equilibrium strategy to improve the found plan. The proposed methodology is polynomial time complete.

The approach presented in the paper is applicable only to the work block model. It should be interesting to extend this work to other domains. The group of robots which move blocks is an agent-based system because there is no communication among the constituent agents. Moreover, when a plan is found (that satisfies non-cooperative Nash equilibrium), it acts as centralized control and the agents only have to execute it, losing their autonomy. Although the plan is improved using a non-cooperative strategy, when the agents execute the general plan (following a Nash equilibrium), they are cooperating in an indirect way.

The computational complexity study presented is applicable only to the Block World.

Acknowledgments

This work was supported by the Polish State Committee for Scientific Research through a grant No. 3 T 11 A 026 28 in 2005 for the first author and by the Silesian University of Technology through a grant in 2005 for the second author.

References

Avriel M., Penn M., Shpirer N. and Witteboon S. (1998): *Stowage planning for container ships to reduce the number of shifts*. — Ann. Oper. Res., Vol. 76, pp. 55–71.

Avriel M., Penn M. and Shpirer N. (2000): *Container ship stowage problem: complexity and connection to the coloring of circle graphs*. — Discr. Appl. Math., Vol. 103, pp. 271–279.

Baral Ch., Kreinovich V. and Trejo R. (2000): *Computational complexity of planning and approximate planning in the presence of incompleteness*. — Artif. Intell., Vol. 122, pp. 241–267.

Basar T. and Olsder G.J. (1982): *Dynamic Noncooperative Game Theory*. — New York: Academic Press.

Belker T., Beetz M. and Cremers. A.B. (2002): *Learning of plan execution policies for indoor navigation*. — AI Comm., Vol. 15, No. 1, pp. 3–16.

Bish E.K., Leong T.Y., Li C.L., Ng J.W.C. and Simchi-Levi D. (2001): *Analysis of a new vehicle scheduling and location problem*. — Naval Res. Logist., Vol. 48, pp. 363–385.

Boutilier C. and Brafman R.I. (2001): *Partial-order planning with concurrent interacting*. — Actions. J. Artif. Intell. Res., Vol. 14, pp. 105–136.

Bylander T. (1994): *The computational complexity of propositional STRIPS planning*. — Artif. Intell., Vol. 69, pp. 165–204.

Fabricant A., Papadimitriou Ch. and Talvar K. (2004): *The complexity of pure Nash equilibria*. — Proc. ACM Symp. *Theory of Computing*, Chicago, pp. 604–612.

Gałuszka A. and Świerniak A. (2002): *Planning in multi-agent environment as inverted STRIPS planning in the presence of uncertainty*, In: Recent Advances In Computers, Computing and Communications (N. Mastorakis and V. Maldenov, Eds.). — Athens: WSEAS Press, pp. 58–63.

Gałuszka A. and Świerniak A. (2003): *STRIPS representation and non-cooperative strategies in multi-robot planning*. — Proc. 15th *European Simulation Symposium (SCS)*, Delft, the Netherlands, pp. 110–115.

Gałuszka A. and Świerniak A. (2003a): *Invertible planning and non-cooperative equilibrium strategies in multi-agent planning*. — Proc. 11th IEEE Mediterranean Conf. *Control & Automation*, Rhodos, Greece, CD-ROM.

Gupta N. and Nau D.S. (1992): *On the complexity of Blocks-World Planning*. — Artif. Intell., Vol. 56, No. 2–3, pp. 223–254.

Howe A.E. and Dahlman E. (2002): *A critical assessment of benchmark comparison in planning*. — J. Artif. Intell. Res., Vol. 17, pp. 1–33.

Imai A., Nishimura E. and Papadimitriou S. (2001): *The dynamic berth allocation problem for a container port*. — Transp. Res., Vol.B 35, pp. 401–417.

Isil Bozma H. and Koditschek D.E. (2001): *Assembly as a non-cooperative game of its pieces: Analysis of 1D sphere assemblies*. — Robotica, Vol. 19, pp. 93–108.

Karacapilidis N.I. and Papadias D. (1998): *A computational approach for argumentative discourse in multi-agent decision making environment*. — AI Comm., Vol. 11, No. 1, pp. 21–33.

Koehler J. and Hoffmann J. (2000): *On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm*. — J. Artif. Intell. Res., Vol. 12, pp. 339–386.

Kraus S., Sycara K. and Evenchik A. (1998): *Reaching agreements through argumentation: A logical model and implementation*. — Artif. Intell., Vol. 1, No. 4, pp. 1–69.

Mc Kinsey J.C. (1952): *Introduction to the Theory of Games*. — New York: Mc Graw Hill.

Mesterton-Gibbons M. (2001): *An Introduction to Game-Theoretic Modelling*. — Providence, RI: American Mathematical Society.

Nilson N.J. (1980): *Principles of Artificial Intelligence*. — Palo Alto, CA: Toga Publishing Company.

Papadimitriou Ch. (2001): *Algorithms, games and the Internet*. — Proc. ACM Symp. *Theory of Computing*, Hersonissos, Greece, pp. 749–753.

Papadimitriou Ch. (2001a): *Theory of the Complexity*. — Warsaw: Polish Scientific Publishers.

Skrzypczyk K. (2005): *Control of a team of mobile robots based on non-cooperative equilibria with partial coordination*. — Int. J. Appl. Math. Comp. Sci., Vol. 15, No. 1, pp. 89–97.

Slaney J. and Thiebaux S. (2001): *Block World revisited*. — Artif. Intell., Vol. 125, pp. 119–153.

Slavin T. (1996): *Virtual port of call*. — New Scientist, No. 15, pp. 40–43.

Smith D.E. and Weld D.S. (1998): *Conformant graphplan*. — Proc. 15th Nat. Conf. Artificial Intelligence, Madison, Wisconsin, USA, pp. 889–896.

Weld D.S. (1999): *Recent Advantages in AI Planning*. — AI Mag. Vol. 20, No. 2, pp. 93–123.

Weld D.S., Anderson C.R. and Smith D.E. (1998): *Extending graphplan to handle uncertainty & sensing actions*. — Proc. 15th Nat. Conf. Artificial Intelligence, Madison, Wisconsin, USA, pp. 897–904.

Wilson I.D. and Roach P.A. (2000): *Container stowage planning: A methodology for generating computerised solutions*. — J. Oper. Res. Soc., Vol. 51, pp. 1248–1255.

Yale Center for Computational Vision and Control (1998): *PDDL – The planning domain definition language*. — Tech. Report CVC TR-98-003/DCS TR-1165.

Zhang Y., Wu Ch. and Bai Y. (2001): *Implementing prioritized logic programming*. — AI Comm., Vol. 14, No. 4, pp. 183–196.

Received: 25 October 2004
Revised: 23 June 2005