Problem Solving in a Simulated Robot World

Mingshu Li[1]

Department of Artificial Intelligence
University of Edinburgh
5 Forrest Hill, Edinburgh EH1 2QL, U.K
mingshu@dai.ed.ac.uk

Abstract

This paper discusses the complexity of, and problem solving for, action planning in a simulated robot world. It firstly puts forward two kinds of typical action planning problems, then presents and proves the intractability of the problems in the simulated world. It also analyses the solving strategies for these problems and introduces some promising approaches to planning problems in simulated robot worlds. The results in this paper can be used as a reference for the study of planning, combinatorial search and other AI problem solving.

1 Introduction

Planning is always the crucial question in combinatorial search and intelligent research in robotics. It means the process of calculating the effect of later actions before carrying out any actions of a problem solving program. In another words, planning is a description of the actions process. Robot planning is the process of automatically generating an actions sequence to make the robot achieve the desired goal, given some original situation.

Limited by the capability and efficiency of image recognition and computing facilities, most of current research on intelligent robotics, especially on problem solving and planning systems, are based on some modelled world. In the literature, there are many famous planning systems in block worlds or simulated robot worlds, such as STRIPS [1], ABSTRIPS [2], PULP-1 [3], ROPES(PULP-24) [4], REFLECT [5], Chang & Wee's system [6], and so on.

Although these have been some traditional problems for many years, a lot of research work has been done on them in recent years [7]. Indeed, many new approaches can be found during research on problem solving in simulated robot worlds; also, those new crops of theories can be tested on the simulated robot problems. This paper will try to establish such a study environment.

2 An Initial Simulated Robot World Model

As most of current research on planning problems are based on some modelled world. Such a model should be neither too complex to be disturbed by extraneous factors, nor too simple to find general planning techniques. Thus, establishing a suitable robot world model is one of the most important techniques in solving robot planning problems.

Firstly, we may suppose an initial robot world model: there are some rooms in one layer or in multi-layers and some objects such as boxes or blocks, switches, windows, tables, chairs, carts and so on, in the rooms. A room may have door(s) or ladder(s) connecting to neighbouring room(s).

[1]On leave from Institute of Software, Chinese Academy of Sciences, P.O.Box 8718, Beijing 100080, P.R.China, *mingshu@ios.ac.cn*.

Every object has its own features, such as its size, color, and initial position and situation. Robots in the world can move here and there freely only if having some passable route, and can do something in the world (for example, carrying a movable object from one position to another position). They will change the original states to the goal states, in the world. In this paper, such a change process is what we call robot problem solving.

To discuss this problem simply and directly, I will present a simplified form of the above-mentioned model: the objects are some blocks with the same size and different signs. No block has any special state characteristics except its position; there are one or two robots which can recognize, hold, carry, and place these blocks one by one, and the two robots can work in coordination by communicating with each other. Whether the world is one-dimensional, two-dimensional or three-dimensional space, the position of each block or robot can be expressed by a coordinate. Fig 1 gives such an example in a two-dimensional (2-D) simulated dual robot world.

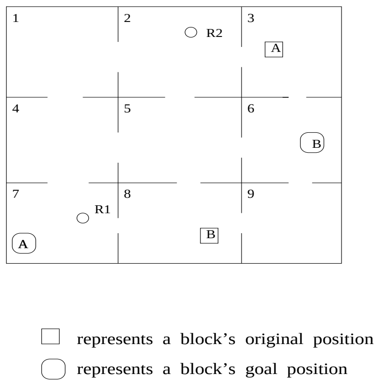


Figure 1: A robot planning problem

A lot of work about need to be done on planning analysis [8] [9]. This paper is an attempt to examine different methods of solving the problem. In this paper, those planning problems involving only a single robot, we shall call them general robot planning problems; and those planning problems involving dual robots, we shall call cooperative robot planning problems. Thus, let us discuss problem solving for general and cooperative robot planning problems in the modelled world.

3 Solving General Robot Planning Problems

3.1 General robot planning problem and its intractability

Suppose there exists one robot and n blocks in the modelled world. The robot will move the n blocks from some original state to some goal state, according to some order of moving blocks, i.e., there exists a function f,

$$f: \ [1 \cdots n] \rightarrow [1 \cdots n]$$

It means some order of moving blocks (i.e., tasks assignment order) can be achieved by the function f based upon some planning algorithm. Then $f(i)$ represents the block moved in the order i. To unifiedly describe the blocks and the robot, we expands the function f as a function from $\{0, 1, \ldots, n+1\}$ to itself, i.e.,

$$f: \ [0 \cdots n+1] \rightarrow [0 \cdots n+1]$$

in which, $f(0) = 0$ represents the robot when it begins moving from its original position; $f(n+1) = n+1$ represents the robot when it reaches its goal position.

INSTANCE: In the simulated robot world, there is a finite set $BO = \{b_1, b_2, \dots, b_n\}$ of blocks, in which each block can take a coordinate as its only sign. A distance $d(b_i, b_j) \in Z^+$ (where Z^+ denotes the positive integers), for each pair of blocks $b_i, b_j \in BO$, represents the distance that the robot carries block b_i from its original position to its goal position, then goes to the original position of block b_j. It also represents the work load, $W(b_i)$, of the robot moving the block b_i. There is a limit $B \in Z^+$.

QUESTION: Can the robot finish the task of moving all the blocks and its work load W will not exceed the limit B, that is, there exists a rank order (moving order) of blocks BO $\langle b_{f(1)}, b_{f(2)}, \dots, b_{f(n)} \rangle$, such that

$$W = \sum_{i=1}^{n+1} W_{b_i} = [\sum_{i=1}^{n-1} d(b_{f(i)}, b_{f(i+1)})] + d(b_{f(n)}, b_{f(n+1)}) \leq B$$

in which, $d(b_{f(n)}, b_{f(n+1)})$ represents the distance from the goal position of the last block to the goal position of the robot itself.

THEOREM 1: GENERAL ROBOT PLANNING PROBLEM in the simulated world is NP-complete.

PROOF: It is easy to see that GENERAL ROBOT PLANNING PROBLEM $\in$ NP, since a nondeterministic algorithm need only guess a moving order of BO and check in polynomial time whether the sum of the distances of moving blocks is smaller than the limit B.

We transform TRAVELING SALESMAN (TSP) [10] to this problem.

Let the finite set $C = \{c_1, c_2, \dots, c_n\}$ of cities, given distance $d(c_i, c_j) \in Z^+$ for each pair of cities $c_i, c_j \in C$, and a limit $B' \in Z^+$, constitute an arbitrary instance of TSP. We must construct a finite set $BO = \{b_1, b_2, \dots, b_n\}$ of blocks, given distance $d(b_i, b_j) \in Z^+$ for each pair of blocks $b_i, b_j \in BO$, and a limit $B \in Z^+$, such that the work load of the robot when it completes the task of moving all the blocks, according a rank order (moving order) of blocks of BO, $\langle b_{f(1)}, b_{f(2)}, \dots, b_{f(n)} \rangle$, satisfying

$$[\sum_{i=1}^{n-1} d(b_{f(i)}, b_{f(i+1)})] + d(b_{f(n)}, b_{f(n+1)}) \leq B \tag{1}$$

if and only if there exists a rank order (traveling order) of cities of C, $\langle c_{g(1)}, c_{g(2)}, \dots, c_{g(n)} \rangle$, satisfying

$$[\sum_{i=1}^{n-1} d(c_{g(i)}, c_{g(i+1)})] + d(c_{g(n)}, c_{g(n+1)}) \leq B' \tag{2}$$

We will construct the set BO by the set C directly, if we may find the corresponding relations one by one, between the n cities in C and the n blocks in BO, and between the distances for each pair of cities in C and the distances for each pair of blocks in BO. In fact, it is easy for us to find such corresponding relations: firstly, we can use the cities in C to construct the blocks in BO; secondly, we can use the distances for each pair of cities in C to construct the distances for each pair of blocks in BO. In addition, we need find a special element b_{n+1} for BO to correspond the goal position of the robot. Thus,

$$BO \;=\; \{b_i : \; b_i = c_i \wedge c_i \in C\} \bigcup \{b_{n+1}\} \tag{3}$$

$$d(b_i, bj) = d(c_i, cj) \tag{4}$$

in which, $i \neq jand1 \leq i, j \leq n$.

To prove this is really a transformation, firstly suppose there exists a rank order of cities of C $\langle c_{g(1)}, c_{g(2)}, \ldots, c_{g(n)} \rangle$, satisfying the formula (2), the problem will be how we can find a rank order (moving order) of blocks of BO $\langle b_{f(1)}, b_{f(2)}, \ldots, b_{f(n)} \rangle$, satisfying the formula (1).

Let $f \Leftrightarrow g$, and we can know the corresponding relations one by one in C and BO, thus according to the formula (3) and (4),

$\forall c_{g(i)}, c_{g(j)} \in C$, a distance will be given as $d(c_{g(i)}, c_{g(j)}) \in Z^+$, representing the traveling distance between the two cities $c_{g(i)}$ and $c_{g(j)}$. Correspondingly,

$\exists b_{f(i)}, b_{f(j)} \in BO$, a distance will be given as $d(b_{f(i)}, b_{f(j)}) \in Z^+$, and $d(b_{f(i)}, b_{f(j)}) = d(c_{g(i)}, c_{g(j)})$, representing the moving distance the robot carries block $b_{f(i)}$ from its original position to its goal position, then goes to the original position of block $b_{f(j)}$. The two above-mentioned distances are equal.

In addition, for $c_{g(n)}, c_{g(1)} \in C$, there exists $d(c_{g(i)}, c_{g(j)}) \in Z^+$ which represents the distance the traveling salesman returns to his beginning city $c_{g(1)}$ from the city $c_{g(n)}$. Correspondingly, for $b_{f(n)}, b_{f(n+1)} \in BO$, there exist $d(b_{f(n)}, b_{f(n+1)}) \in Z^+$ which represents the distance the robot carries the block $b_{f(n)}$ from its original position to its goal position, then goes to the robot goal position $b_{f(n+1)}$. The two above-mentioned distances are corresponding.

Note that we tacitly approve the distance that the traveling man moves to find his beginning city $c_{g(1)}$ is the same as the distance that the robot moves to find its first movable block $b_{f(1)}$.

Thus, only let $B = B' = max\{B, B'\}$, i.e., both the two limits get the bigger one between the original B and B', then the formula (1) will be true if the formula (2) is true.

Conversely, suppose there exists a rank order of blocks of BO, $\langle b_{f(1)}, b_{f(2)}, \ldots, b_{f(n)} \rangle$, satisfying the formula (1), we need to find a rank order of cities of C, $\langle c_{g(1)}, c_{g(2)}, \ldots, c_{g(n)} \rangle$, satisfying the formula (2).

Let $g \Leftrightarrow f$, then

$\forall b_{f(i)}, b_{f(j)} \in BO$, and $d(b_{f(i)}, b_{f(j)}) \in Z^+$, $\exists c_{g(i)}, c_{g(j)} \in C$, and $d(c_{g(i)}, c_{g(j)}) \in Z^+$, satisfying $d(c_{g(i)}, c_{g(j)}) = d(b_{f(i)}, b_{f(j)})$.

For $d(b_{f(n)}, b_{f(n+1)}) \in Z^+$, correspondingly there exists $c_{g(n)}, c_{g(1)} \in C$.

Thus, also only let $B' = B = max\{B, B'\}$, then the formula (2) will be true if the formula (1) is true.

As there is a corresponding relation between the set C and set BO, it is easy to see that the above-mentioned transformation can be finished in polynomial time. □

3.2 Solving strategies for general robot planning problems

THEOREM 1 are presented and proved in a supposed robot world. However, in the world model, the general robot planning problem can be abstracted as a simplified summing up problem of the distances which the robot moves the blocks in the world. Thus, it is the most simplified and abstracted model, in which a thorough investigation can be made for general robot problem solving. The proof process of the theorem describes the solving process of the general robot planning problems. Certainly, it is still difficult for computers to solve these with high efficiency. In fact, now that we have presented and proved the intractabilities of general robot planning problems in the modelled world, we can not find an efficient polynomial algorithm to solve those problems. We have to try other strategies for robot planning problems.

Firstly, we may image some special cases. The discussion in THEOREM 1 hides such a presupposition that the robot did not decide the order of moving blocks in advance, i.e., the solving process is nondeterministic. Without this presupposition, more efficient solving tactics may be achieved according to the requirements in the actual task environment. Let suppose a special case for THEOREM 1, the work load of the robot will be counted only if it moves a block; the work load will not be counted if it moves itself. It also has some actual significances. Then the GENERAL ROBOT PLANNING PROBLEM is not NP-complete and will be easy for computers to solve with high efficiency.

THEOREM 2: GENERAL ROBOT PLANNING PROBLEM in the simulated world can be solved in polynomial time, if the work load of the robot may not be counted when it moves itself.

PROOF: Let the distance of the robot moving represents the work load of the robot planning. As all the work loads of the robot in the simulated world is only depended on the original position and the goal position for each block, and irrelevant with the order of the blocks processing. Thus, let $f(i) = i$,

$$W_{min} = min_f \sum_{i=1}^{n} d_{f(i)} = \sum_{i=1}^{n} d_i \leq cn \tag{5}$$

in which $d_{f(i)}$ (or d_i) represents the distance travelled in moving block b_i from its original position to its goal position (it does not include the distance the robot moves itself from some position to the original position of block $b_{f(i)}$ (or b_i)), $c = max_{1 \leq i \leq n}\{d_i\}$.

From the formula (5), the sum of the work loads of the robot will be lower than or equal to the product of the longest distance for all the blocks from their original position to their goal position, and the number of blocks. It is just a linear function of the variable n. □

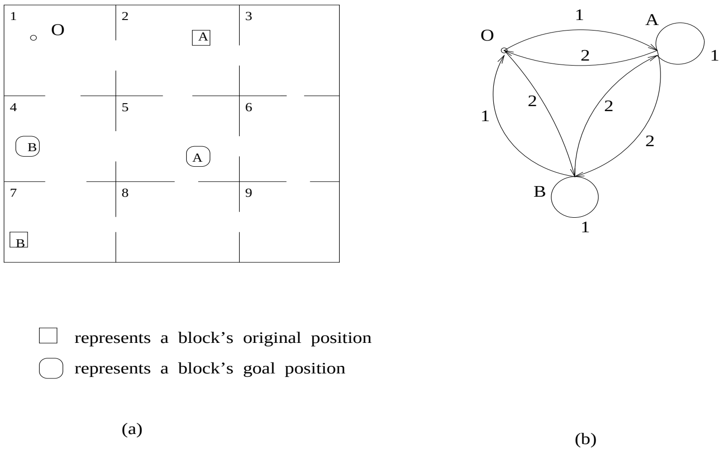


Figure 2: Planning is transformed to searching

Now, we discuss how we can decide the order of moving blocks for a single robot. We may think the distance from the original position to the goal position of a block is determinist; but the distance from the goal position of one block to the original position of another block is nondeterministic. Processing the nondeterministic efficiently will depend on two factors: finding the shortest path from the goal position of finished block to the original position of another movable block; using the enabling-condition interactions which have side-effect [11]. In fact, we may transform the general robot planning problem to a combinatorial search problem, as shown in Fig 2. In Fig 2 (a), the planning task is to find an action sequence to make the robot starting from O position, then moving the block A from $room-2$ to $room-5$, and block B from $room-7$ to $room-4$, respectively; in Fig 2 (b), the searching task is to find a shortest path starting from the source vertex O and passing the vertex A and B and their reflexive paths (a reflexive path represents the distance of a block from its original position to its goal position), respectively, then backing to the source vertex O in the directional weighted graph. It is easy to see that they form a set of related pairs. Thus, a planning problem can be taken as a problem for finding the shortest path or a TSP problem. We will discuss in detail about how to find the shortest path.

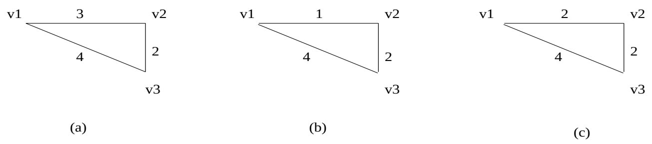


Figure 3: Three example in shortest path problems

Dijkstra Algorithm is one of the most popular algorithms for finding the shortest path. However, it can only give the last result of the shortest path, but not record the intermediate results. Whereas those intermediate results are usually valuable to plan. In Fig 3, *Dijkstra Algorithm* can easily solve all the three problems. But in problem (c), it can only give the length of the shortest path from $v1$ to $v3$ is 4, and can not distinguish that there are two shortest paths, i.e., $v1 \rightarrow v2 \rightarrow v3$ and $v1 \rightarrow v3$.

Here, I present a finding shortest path algorithm, names as *Comparing and Storing Algorithm*, which is an improvement on the *Dijkstra Algorithm*. Its basic principle is that of comparing and storing after searching one level, and then keeping the shortest subpath in this search, and the last vertex reaching this vertex in the shortest subpath, according to a similar algorithm, the *Broad-First Algorithm*. It can be described as follows:

Comparing and Storing Algorithm

input: weighted matrix W, let $w(i,j) = \infty$ when there is no edge to connect vertex i and j, and $w(i,i) = \infty$;

output: dist(i) and dist(i,j,t), $1 \leq i, j, t \leq n$, in which both dist(i), dist(i,j,t) represent the shortest path (distance) from the source vertex (root) s to vertex i, j is the last vertex of i which on the shortest path, t represents the level (edge) number from s to i.

begin

$VL_0 \leftarrow V;\ VL_1 \leftarrow \{s\};\ VL_2 \leftarrow \emptyset;\ t \leftarrow 1;$

$dist(s) \leftarrow 0;\ dist(i) \leftarrow \infty (1 \leq i \leq n, i \neq s);$

while $VL_0 \neq \emptyset$ **do**

  for *each* $(u \in VL_1) \bigwedge (\langle u,v\rangle \in E)$ **do**

    $dist(v,u,t) \leftarrow dist(u) + w(u,v);$

    if $v \in VL_0$ **then** $VL_2 \leftarrow \{v\}$

  endfor

  for each $(v \in VL_2) \bigwedge (u \in VL_1)$ **do**

    $dist(v) \leftarrow min\{dist(v,u,t), dist(v)\}$

  endfor

  $VL_0 \leftarrow VL_0 - VL_1;\ VL_1 \leftarrow VL_2;\ t \leftarrow t+1$

endwhile

endbegin

Using the *Comparing and Storing Algorithm* to solve the problem (c) in Fig 3, the output will be as follows:

$$\begin{aligned} dist(3) &= 4 \\ dist(3,1,1) &= 4 \\ dist(3,2,2) &= 4 \end{aligned}$$

It will give the best plan by finding the shortest path (the distance is 6) to the problem in Fig 2,

$$O \to A \to B \to O, \ i.e.$$

$$1 \to 2 \to 5 \to 4 \to 7 \to 4 \to 1$$

the other possible plan is (the distance is 8),

$$O \to B \to A \to O, \ i.e.$$

$$1 \to 4 \to 7 \to 4 \to 1 \to 2 \to 5 \to 2 \to 1$$

Let us analysis the correctness and complexity of *Comparing and Storing Algorithm*.

THEOREM 3: *Comparing and Storing Algorithm* can correctly find the shortest paths from a given vertex to all the other vertexes and its time complexity is $O(n^2)$.

PROOF: As the search strategies in the Algorithm are broad-first on OR-graphs, it must get the shortest path. Thus, the Algorithm is correct.

The line 1, 2 and 11 conduct valuation operation on the time complexity $O(n)$. The for cycle in line 4-7 and in 8-10 hold a dominant position in the $while$ cycle in line 3-12. The for cycle in line 4-7 need $O(n)$ steps, and the for cycle in line 8-10 will not exceed $O(n)$ steps in general. The $while$ cycle will run at most $O(n)$ times. Thus, the complexity of the Algorithm is $O(n^2)$.

Note that, we tacitly approve the time spent by plus operation is the same or similar as that by comparison operation. □

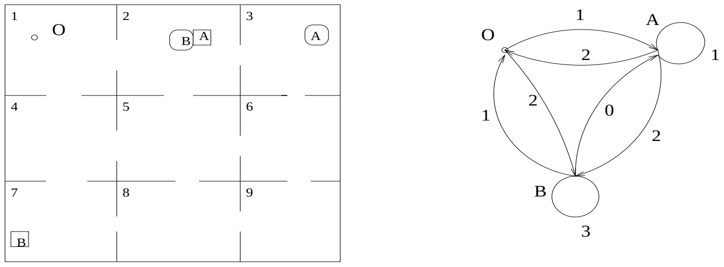
It is easily parallelizable for *Comparing and Storing Algorithm*.

THEOREM 4: The time complexity is $O(n)$ and the processor complexity is $O(n)$, for *Parallel Comparing and Storing Algorithm* in SIMD-CREW PRAM (Single Instruction Stream Multiple Data Stream, Concurrent-Read and Exclusive-Write Parallel Random Access Machine) with n processors.

PROOF: It is clear that the operations in line 1, 2 and 11 will be finished in $O(1)$. The for cycle in line 4-7 will be also finished in $O(1)$ as there are n processors to conduct parallel valuation operation at the same time. The for cycle in line 8-10 can be finished in $O(1)$, though it has two variables (if in the extreme, the for cycle in line 8-10 run $O(n)$ times, then the $while$ cycle will only run $O(1)$ times). And the $while$ cycle in line 3-12 will run at most $O(n)$. Thus, the time complexity is $O(n)$ and the processor complexity is $O(n)$. □

Let us discuss another planning example of transformed searching shown in Fig 4.

In Fig 4 (a), the planning task is to find an action sequence to make the robot starting from O position, then moving the block A from $room-2$ to $room-3$, and block B from $room-7$ to $room-2$, respectively; in Fig 4 (b), the searching task is to find a shortest path starting from the source vertex O and passing the vertex A and B and their reflexive paths respectively, then backing to the source vertex O in the directional weighted graph. It is similar to the problem in Fig 2. However, the best plan (the distance is 8) will be,



(a) (b)

Figure 4: An enabling-condition interaction example

$$O \to B \to A \to O, \ i.e.$$

$$1 \to 4 \to 7 \to 4 \to 1 \to 2 \to 3 \to 2 \to 1$$

the other possible plan is (the distance is 10),

$$O \to A \to B \to O, \ i.e.$$

$$1 \to 2 \to 3 \to 2 \to 1 \to 4 \to 7 \to 4 \to 1 \to 2 \to 1$$

In Fig4, why is the plan, where the robot first moves block A which is nearest to it (according to some direct ideas, it should do this), not as good as having the robot first move block B which is farther from it? It is just an enabling-condition interaction which has side-effect, i.e., the robot reaches the original position of block A at the same time finishing to move block B to its goal position. There may be two strategies to use the side-effect to make a better plan: (1) taking concurrent or parallel search technologies to compare and select from as many plan solutions as possible; (2) introducing some heuristics knowledge (e.g., in the problem in Fig 4, we can add a restricted condition in the searching algorithm: if the distance from the goal position of a block A to the original position of another block B, is shorter than the distance from the current position of the robot to the original position of the block B, then let the robot firstly move the block A).

3.3 A knowledge-based robot problem solver KBROPS

The KBROPS system mainly consists of five component parts [12]: (1) the knowledge base: storing the planning and problem solving knowledge and tactics which can be termed as domain knowledge, task knowledge and plan knowledge; (2) the interpreting & learning module: contacting with the external knowledge sources and automatically/half-automatically building, adding and renewing knowledge; (3) the multilevel planner: structured as three-level hierarchy of mission planning module, path planning module and local planning module; (4) the system monitor and the control strategy: coordinating these components working according to the strategy of search and match, selection and execution; (5) the sensors and executive installation: getting in touch with an external world based on the functions of information extraction, interpretation and action execution. The relations among those component parts can be shown as Fig 5.

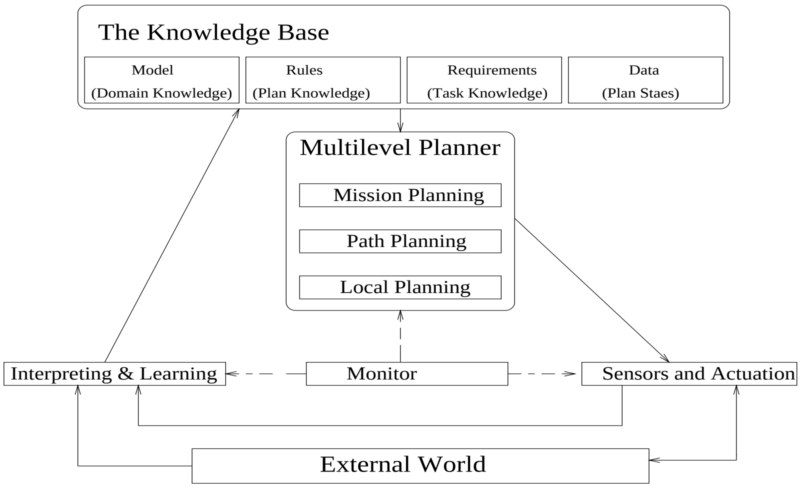


Figure 5: Relations among the component parts in KBROPS

At first, either the programmer or the KBROPS system itself transforms the information about the robot working environment into the domain knowledge; the requirements about the mission into the task knowledge; the methods of solving problems into the tactics of mission resolving, path selecting and action executing (i.e., the plan knowledge); and the initial data will be systematized to constitute the initial data base. Thus the knowledge base used to solve the actual problem have come into being.

Using knowledge is the biggest feature of KBROPS. According to the requirements of task knowledge, the mission planning module depends on the domain knowledge and the mission resolving tactics to reason, and finishes the mission planning under the set binding conditions; the path planning module uses the path knowledge such as the path selecting and action executing tactics to produce the positions representation of the planning path from the geographic goals and constrains, and processes the representation of position-to-position into the representation of symbol path which can be executed by the lower module; the local planning module transforms the representation of symbol path into the action sequence using the heuristic knowledge of actions execution, then uses the knowledge of actions execution, of environment understanding, of diagnosis and of maintenance in the actual processing to directly control and drive the executive installation.

In the planning process, after any planning knowledge is used or any action is carried out, the data in the data base which indicates the planning process and state will be changed until the data state required by the problem (the goal data base) is reached. These interrelated actions are arranged into a robot action sequence from the initial state to the goal state, and this is the planning result. In the actual executive process, the system should be able to dynamically repair or adjust the plan to improve the planning performance according to the executive situation.

Implemented by OPS83 and C programming language, KBROPS has special considerations for general robot planning problems in the simulated world as follows:

In the mission planning sub-process, the complex problems will be resolved into a simple and exercisable sub-problems sequence, and the working order of finishing the planning task also decided. It divides the task of moving a block into two part: one is the robot moves itself from its current position to the position where the block exists; the other is the robot moves the block from its initial position to its goal position.

When the mission planning sub-process begins, the robot firstly calls the sub-procedure of path searching and find the block most nearing the robot, making moving the block the first task and storing the sub-task sequence. Supposing this task has been finished, the system then modifies the robot's current position to the block's goal position to find the next nearest block. Continuing in this way, after all the blocks have been processed, the system finishes its mission planning sub-process and enters its path planning sub-process.

The path planning sub-process mainly decides which path will be selected when the robot moves from the initial room to the goal room. The shortest path can be found in the path-table provided by the system. The path stored in the path-table must be a shortest path. If the path from the initial to the goal room does not exist in the path-table, the system will call the sub-procedure of path searching to find the shortest path.

The sub-procedure of path searching is implemented by C programming language and is the most important external procedure in KBROPS. Using ingenious algorithm and architecture, it can efficiently find the shortest path from one room to another room in the simulated world model. In the whole process of robot planning, the path from one room to another room will be searched only one time and then be stored in the path-table, including the rooms the path passes. So the KBROPS system has some learning capability and only limited path searching will be needed. For path searching always being a bottleneck, the KBROPS system can improve the efficiency greatly by using the techniques of path-table, especially in the complex problems.

The local planning sub-process will decide the suitable actions sequence, based on the sub-path sequence obtained by the path planning module.

4 Solving Cooperative Robot Planning Problems

4.1 Cooperative planning problem and its intractability in a 2-D simulated robot world

Suppose there exists two robots and n blocks in the modelled world. The two robots can work in coordination by communicating with each other, and move the n blocks from some original state to some goal state. They will fulfil almost same work load.

INSTANCE: In a simulated robot world, a finite set $BO = \{b_1, b_2, \ldots, b_n\}$ of blocks, in which each block can take the coordinate as its only sign. A distance $d(b_i, b_j) \in Z^+$ (where Z^+ denotes the positive integers), for each pair of blocks $b_i, b_j \in BO$, represents the distance that one of the dual robots carries block b_i from its original position to its goal position, then goes to the original position of block b_j. It also represents the work load, $W(b_i)$, of the robot moving the block b_i.

QUESTION: Can the dual robots work in coordination and in balance to finish the planning task of moving all the blocks, that is, there exist subsets of blocks $BO' \subseteq BO$ and $BO'' \subseteq BO$, such that,

$$\sum_{b_i \in BO'} W(b_i) \;=\; \sum_{b_i \in BO''} W(b_i) \;=\; \sum_{b_i \in BO - BO'} W(b_i)$$

THEOREM 5: COOPERATIVE DUAL ROBOTS PLANNING PROBLEM in the simulated world is NP-complete.

PROOF: It is easy to see that COOPERATIVE DUAL ROBOTS PLANNING PROBLEM $\in$ NP, since a nondeterministic algorithm need only guess a subset BO' of BO and check in polynomial time that the sum of the work loads of moving the blocks in BO' is the same as that for moving the blocks in $BO - BO'$.

We transform PARTITION [13] to this problem.

Let the finite set A and given size $S(a) \in Z^+$ for each $a \in A$ constitute an arbitrary instance of PARTITION. We must construct a finite set $BO = \{b_1, b_2, \ldots, b_n\}$ of blocks, and a work load $W(b_i) \in Z^+$ of moving the block b_i for each $b_i \in BO$, such that BO contains a subset BO' ($BO' \subseteq BO$) satisfying

$$\sum_{b_i \in BO'} W(b_i) = \sum_{b_i \in BO - BO'} W(b_i) \tag{6}$$

if and only if A contains a subset A' ($A' \subseteq A$) satisfying

$$\sum_{a \in A'} S(a) = \sum_{a \in A - A'} S(a) \tag{7}$$

Supposing the finite set A is a set with n elements, that is, $A = \{a_1, a_2, \ldots, a_n\}$.

We will construct the set BO by the set A directly, if we may find the corresponding relations one by one, between the n elements in A and the n blocks in BO, and between the size $S(a_i)$ for each $a_i \in A$ and the work load $W(b_i)$ for each $b_i \in BO$. In fact, it is easy for us to find such corresponding relations. Thus,

$$BO = \{b_i : b_i = a_i \wedge a_i \in A\} \tag{8}$$

$$W(b_i) = S(a_i) \tag{9}$$

in which $1 \leq i \leq n$.

To prove this is really a transformation, firstly suppose there exists a subset $A' \subseteq A$ satisfying the formula (7), thus according to the formula (8) and (9), $\sum_{b_i \in BO'} W(b_i) = \sum_{a_i \in A'} S(a_i)$, and $\sum_{b_i \in BO - BO'} W(b_i) = \sum_{a_i \in A - A'} S(a_i)$, where $BO' = \{b_i : b_i = a_i \wedge a_i \in A'\}$. Thus,

$$\sum_{b_i \in BO'} W(b_i) = \sum_{b_i \in BO - BO'} W(b_i)$$

that is the formula (6) is true.

Conversely, suppose there exists a subset $BO' \subseteq BO$ satisfying the formula (6), let $A' = \{a_i : a_i = b_i \wedge b_i \in BO'\}$, using the same method can conclude,

$$\sum_{a \in A'} S(a) = \sum_{a \in A - A'} S(a)$$

that is the formula (7) is true.

As there is a corresponding relation between the set A and set BO, it is easy to see that the above-mentioned transformation can be finished in polynomial time. $\square$

THEOREM 5 is presented and proved in a supposed 2-D robot world. However, in the world model, the cooperative planning problem can be abstracted as a simplified distribution problem of the distances which the robots move the blocks in the world. Thus, it is also the most simplified and abstracted model, in which a thorough investigation can be made for cooperative robot problem solving. The proof process of the theorem also describes the solving process of the cooperative planning problems. Similarly, it is difficult for computers to carry out cooperative solving with high efficiency.

4.2 Solving strategies for cooperative robot problems

Like THEOREM 1, the discussion of THEOREM 5 also hides a presupposition that the work load of moving each block almost always is different. In fact, if we take moving a block, no matter what its distance is, as a task unit, i.e., moving all the blocks will spend the same work load, then the COOPERATIVE DUAL ROBOTS PLANNING PROBLEM is not NP-complete and will be easy for computers to solve with high efficiency. Such a supposition also has some actual significances.

THEOREM 6: COOPERATIVE DUAL ROBOTS PLANNING PROBLEM in the simulated world can be solved in polynomial time, if we take moving a block as a task unit.

PROOF: As we have taken moving a block, no matter what its distance is, as a task unit, the total work load of robots in the simulated world is only dependent on the number of blocks. Thus,

$$\sum_{b_i \in BO'} W(b_i) = \sum_{b_i \in BO-BO'} W(b_i) = \frac{1}{2}n \tag{10}$$

The formula (10) is just a linear function of the variable n. □

Now, let us discuss how can decide the task partition of moving blocks for the dual robots. Undoubtedly, we have to introduce more special strategies for solving cooperative robot planning problems than general robot planning problems.

Cooperative robot problem solving or cooperative robot planning means multiple robots work in coordination and keep in step to solve a complex problem together. It is necessary for complex tasks, since single robot with limited function can not achieve the tasks. The key point in cooperative robot problem solving is centered on coordination and cooperation [14].

The approaches to cooperative robot problem solving can be classified into two types [15]: the *centralized* approach is to plan the action for all the robots using a centralized planner; the *decentralized* approach is to distribute the planner among the robots. In a centralized system, one master robot exists and other robots obey the master. The master robot is required to have many functions and if it breaks down, the task can not be achieved. On the other hand, in a decentralized system, each robot has a function and since plural robots work cooperatively for the one task, it is easy to deal with various tasks. In practice, it is difficult to achieve the decentralized system, because interaction among robots uses extra amounts of public resources which are always limited. Therefore, competition among the robots may occur when robots use the public resource. It may cause collisions and deadlock states among the robots in a local area. In order to avoid the competition, it is necessary for the robots to communicate locally at least and to coordinate among themselves. And if the robots are not properly coordinated, the efficiency will be reduced. So this paper will integrate the two approaches.

Cooperative robot planning algorithms and programming languages or tools have also been proposed in recent years. Most of them have dealt with the cooperative control of flexible manipulators or fingers. In this subsection, three general cooperative robot planning problem examples will be given and issued to introduce an experimental cooperative planning algorithm for dual robots in a simulated robot world. Certainly, it can be extended to a cooperative multiple robot planning algorithm easily based on the same idea. The cooperative and coordinated mechanism is just what will be aimed at in this paper.

To solve these cooperative robot planning problems, two tasks must be fully considered: (1) when there are multiple routes from one position to another position which may be selected by a robot, how to choose the shortest route; (2) when facing multiple subtasks to be completed, how to distribute these subtasks among the two robots to make them work in coordination, in balance and at the lowest work load. The former is a shortest route problem and may be found in last section; the later is just a cooperative planning problem and will be discussed in this section.

A direct idea is distributing the moving tasks to $robot-1$ firstly. As soon as $robot-1$ finds a suitable block to move, $robot-2$ begins to find a suitable block to move from the remainder block set; and so on and so forth, until all the moving tasks have been distributed. Such a idea can be implemented by *Planning Algorithm 1.*

Planning Algorithm 1

begin

s = number of blocks;

p = 1

while s $\neq$ 0

do let $robot - p$ finding the nearest block from it to move;

modifying the current room of $robot - p$ as the goal room of the block it moving;

s = s - 1;

p = 3 - p

enddo

endbegin

Planning Algorithm 1 is a "greed" algorithm indeed: each robot wanting to find the nearest block to move. For the problem given in Fig 1, it is easy to see, according to *Planning Algorithm 1*, $robot - 1$ will move block B from $room - 8$ to $room - 6$ and $robot - 2$ will move block A from $room - 3$ to $room - 7$. *Planning Algorithm 1* is the simplest one to implement by programming languages or tools, but often can not get a best solution: as a block which is near to a robot may be much nearer to the other one. For the example shown in Fig 6, $robot - 1$ will firstly find block B to move if according the order of right, up, left and down, then $robot - 2$ has to move block A. It is clear that it will be better if let $robot - 1$ move A and $robot - 2$ move B. As an improvement, in *Planning Algorithm 2*, if the block found by a robot is much nearer to the other robot, let the other robot move this block.

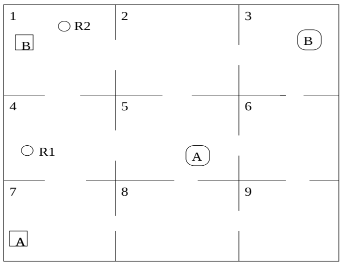


Figure 6: Another cooperative planning problem

Planning Algorithm 2

begin

s = number of blocks;

p = 1

while s $\neq$ 0

do

if $robot-p$ have a block $\langle A \rangle$ awaiting it to move

  then let $robot-p$ move $\langle A \rangle$;

    modifying the current room of $robot-p$ as the goal room of $\langle A \rangle$;

    s = s - 1;

    p = 3 - p

  else let $robot-p$ finding the nearest block from it $\langle B \rangle$ to move;

    if $\langle B \rangle$ is nearer to the other robot and that robot have not a block awaiting it to move

      then let the other robot $robot-(3-p)$ in thestatus ready to move $\langle B \rangle$

        let $robot-p$ finding the nearest block from it, except $\langle B \rangle$, $\langle C \rangle$ to move;

        let $robot-p$ in the status ready to move $\langle C \rangle$

      else let $robot-p$ in the status ready to move $\langle B \rangle$

    endif

endif

enddo

endbegin

When one of the two robots uses *Planning Algorithm 2* to plan, it tries to consider the other robot. It can get a best solution in solving the problems in Fig 1 and Fig 6. However, it will lead to just the opposite in solving another problem in Fig 7: the distance (number of rooms) from block B to $robot-1$ is 3, to $robot-2$ is 2, if according to *Planning Algorithm 2*, let $robot-2$ move B, then $robot-1$ has to move A (the distance will be 4).

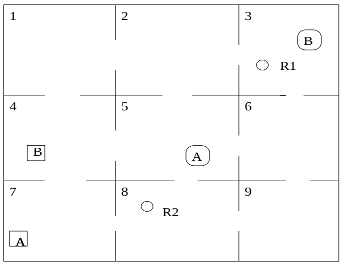


Figure 7: The third cooperative planning problem

Both of the two above-mentioned algorithms only pay attention to finding a suitable task for a single robot, but ignore the cooperation between the two robots. Although the later one tries to consider the other robot, it usually only attends to one thing and loses sight of the other. Thus, I will now present a *Cooperative Planning Algorithm.* The basic idea in this algorithm can be summarized as: each robot tries to find the block which is nearer to it and further from the other robot. It fully embodies a cooperative planning strategy.

Cooperative Planning Algorithm

begin

s = number of blocks;

```
p = 1
while s ≠ 0
do
    d = 0
    loop:
    if robot − p finding block(s) with a distance d
        then
            if only a movable block ⟨A⟩
                then let robot − p move ⟨A⟩;
                    modifying the current room of robot − p as the goal room of ⟨A⟩;
                    s = s - 1;
                    p = 3 - p
                    d = 0
                    goto loop
                else let robot − p finding the furthest block from the other robot robot −
                  (3 − p), ⟨B⟩ to move;
                    modifying the current room of robot − p as the goal room of ⟨B⟩;
                    s = s - 1;
                    p = 3 - p
                    d = 0
                    goto loop
            endif
        else
            d = d + 1;
            goto loop
    endif
enddo
endbegin
```

In *Cooperative Planning Algorithm*, each robot will firstly find the block whose distance is d (the initial value is 0, and will be added 1 each time). If the robot only finds one block at the distance d, the block is just what the robot should move. If the number of the robot finding blocks at this distance is more than one, a conflict has happened in the tasks distribution. Let the robot choose the furthest block from the other robot to move. Then let the other robot begin to choose a block to move. And so on and so forth, until all the moving tasks have been distributed. Some heuristic knowledge and strategies can be introduces in this algorithm to solve more complex planning problems.

It should be pointed out that only cooperative dual robot planning problems have been cited in this paper merely for easy description. In fact, solving multiple robot planning problems will based on the same principle of the *Cooperative Planning Algorithm*.

How to implement the *Cooperative Planning Algorithm* by a programming language or tool? A suitable programming environment supporting cooperative problem solving is necessary. The rule-based method is the most important problem solving tool. It uses a deductive approach to deduce solutions from a set of rules. This method makes use of the existing domain theory and is more effective than other reasoning approaches when the theory of the problem domain can be well defined. However, traditional rule-based systems are not suitable for distributed and cooperative environments. Although a lot of concurrent rule-based systems have been developed and can be classified into three categories [16]: synchronous parallel rule-based systems, asynchronous parallel rule-based or distributed rule-based systems and multiagent rule-based systems, people feel that rule-based systems are not suited to dynamically handle real-world problems, since rule-based systems have been utilized for describing static knowledge. So a thorough improvement to traditional rule-based systems should be made. Such a new idea supporting rule-based cooperative problem solving, called *Function Module System* (FM_S for short) and the programming language FM_S-1 have been presented [17].

4.3 A cooperative robot problem solver KBCOPS

A rule-based cooperative dual robots problem solver has been implemented in the concurrent environment of FM_S-1, called it KBCOPS (Knowledge-Based COoperative robot Problem Solver).

KBCOPS based on FM_S-1 consists of five function units: planning control module, ROBOT-1 planning module and action module, ROBOT-2 planning module and action module, shown as Fig 8. (1) Planning control module. It mainly coordinates planning and execution between the two robots, and decides how to assign the tasks to get an optimum result. (2) Planning modules. The two robots have the same planning module. They store in their database the state messages, environment messages of the modelled world and solving task messages. Each module can carry out task planning and path planning, under the control of planning control module. (3) Action modules. The two robots also have the same action module. They will finish the task of moving blocks, according to the path messages from planning modules and their own state messages. Each robot or each functional robot module can be taken as an intelligent agent.

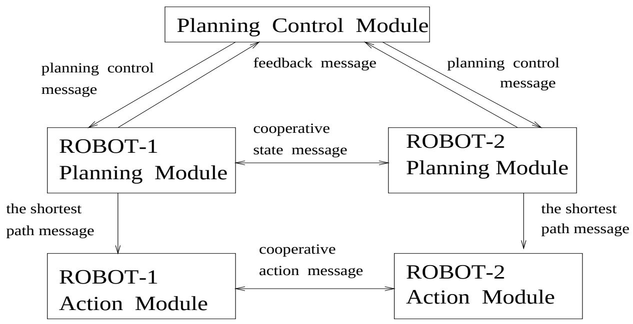


Figure 8: The architecture of KBCOPS

The planning control module will take the role of master function unit to achieve centralized control. On the other hand, the planning modules and the action modules will achieve decentralized coordination.

Communication and cooperation is one of the key problems in cooperative problem solving. The difficulties for cooperative problem solving are: (1) Conflict avoiding. Conflicts in cooperative robot problems can appear in the two aspects of time and space. In this paper, avoiding the conflicts in time means one robot can not repeat the same action or task at any time which has been finished by the other robot. Avoiding the conflicts in space means the two robots can not carry out action-level sub-tasks in the same position, at the same time (e.g., the two robots can not hold the same block in some position at the same time, or place the blocks in their arms in the same position at the same time). (2) Work loads balancing and optimizing. When facing a complex task, decomposition and assignment of the task will directly affect the results of cooperative problem solving. The best solution can only be addressed after all the possible solutions have been obtained. Otherwise, it will waste huge amounts of time and space and be of no use of planning and problem solving. A suitable technique is adding the heuristic knowledge about task decomposition and assignment, and strengthening the cooperative abilities between the two robots as far as possible.

To solve both of the above-mentioned difficulties will need the communication and cooperation mechanism of FM_S-1. For FM_S-1 providing two kinds of communication modes, KBCOPS implements two kinds of cooperative work methods correspondingly: (1) "Waiting in synchronous communication". When a robot function unit needs some cooperative messages from other robot function unit(s), it will send a synchronous request message and will stop its current execution and enter into the active waiting state, until receiving the synchronous reply messages. (2) "Enquiring in asynchronous communication". When a robot function unit has finished some work with cooperative meaning, it will send an asynchronous message to the relevant robot function unit(s) and will continue its current execution. When it needs some cooperative messages from other robot function unit(s), it will enquire from the data buffers to get the messages, sent by other robot function units in advance. Each method has its own strong points and weak points: the former loses some efficiency in the waiting time, but it has clear logical relations so that it seldom makes a mistake; the later can achieve high efficiency for its low communication cost, but it is difficult to manage its data buffers for a complex problem and maybe cause the data loss or confusion. KBCOPS integrates the two kinds of methods in its cooperation.

For the example shown in Fig 6, we may image that without communication and cooperation, as there is the same distance from ROBOT-1 to block A and B, it is possible to assign ROBOT-1 to move block B according to some search algorithm, in which case ROBOT-2 has to pass at least two rooms to move block A. It is clear that ROBOT-1 should move block A and ROBOT-2 move block B. Such cooperative messages (e.g., avoiding both the robots try to move block A and getting a "beneficial" solving result) are passed and acted upon based on the communication among the robot function units.

5 Some Test Results

5.1 Solution for general robot planning problems

STRIPS [1], ABSTRIPS [2], PULP-1 [3] and ROPES [4] are the representative of early planning systems, hierarchical planning systems, planning systems with learning capability and planning expert systems, respectively. To display the solving strategies mentioned above and the planning performance in KBROPS, firstly we may select the above-mentioned four representative planning systems to make a comparison.

Table 1 Comparison of the world model

Planning systems	Oblect Number				
	Rooms	Doors	Boxes	Others	Total
STRIPS	5	4	3	1	13
ABSTRIPS	7	8	3	0	18
PULP-1	6	6	5	12	27
ROPES	6	7	5	15	33
KBROPS	27	28	10	10	75

The ROPES system was implemented in C-PROLOG on a DUAL-VAX 11/780 computer. The PULP-1 system was executed using interpreted LISP on a CDC-6500 computer, the STRIPS and ABSTRIPS were in partially compiled LISP executing garbage collection on a PDP-10 computer. Whereas the KBROPS system was implemented in OPS83 and C programming language on an ordinary PC/AT computer. The CDC-6500 is estimated, on the average, to be about 8 times faster than the PDP-10. However, due to the advantage of the partial compilation and the exclusion of the garbage collection of PDP-10, the CDC-6500 gains slightly in speed of processing. The DUAL-VAX 11/780 is slower than CDC-6500. The PC/AT is only a microcomputer and much slower than DUAL-VAX 11/780. Certainly, OPS83 and C are faster programming languages. In order to simplify the comparison, this paper treat the five systems using the same computer time unit, and make a direct comparison of them.

Table 1 shows the complexity of the five systems. It is clear that the KBROPS system is the most complex, ROPES and PULP-1 are next, and more complex than ABSTRIPS and STRIPS.

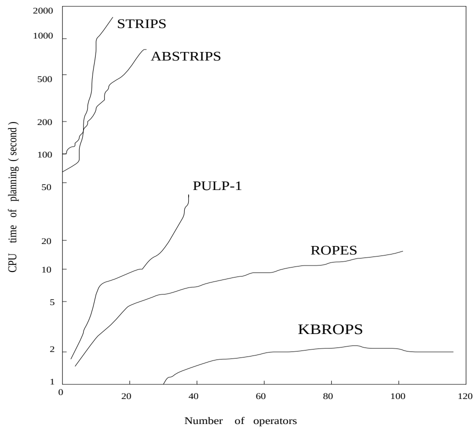


Figure 9: Comparison of planning speed

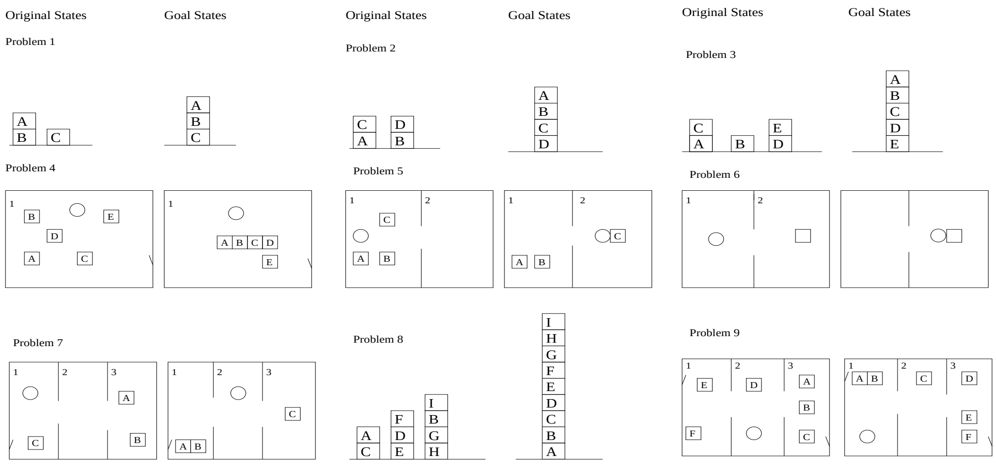


Figure 10: The same nine planning problems

The planning speed of the five systems is plotted in Fig 9 using logarithmic coordinates. From the curves, we can see that the planning speed of PULP-1 is much faster than that of the STRIPS and ABSTRIPS. For instance, in the case of 11 operators, PULP-1 is about 450 times and 82 times faster than STRIPS and ABSTRIPS respectively. For example, in the case of operators 19 and 26, ROPES is 3 times faster than PULP-1. However, we easily find that the planning speed of KBROPS is incomparable. We may take an example: in the case of 53 operators, the planning time of ROPES is 2.950 seconds, and KBROPS is 0.71 seconds (where we need not see the planning result on the monitor). This means KBROPS is about 4 times faster than ROPES. In fact, the KBROPS system is even faster than this, as its computing speed is the slowest among above-mentioned computers.

Table 2 Comparison of the planning results

problems		1	2	3	4	5	6	7	8	9
REFLECT	Steps in plan	6	8	10	10	4	7	34		
	Time(seconds)	1.56	4.85	7.93	9.99	1.14	1.29	86.87		
Chang & Wee	Steps in plan	6	8	10	20	4	6	42	24	80
	Time(seconds)	1.85	2.51	4.1	4.56	1.1	1.9	17.1	11.2	26.9
KBROPS	Steps in plan	14	16	20	21	7	6	58	40	110
	Time(seconds)	0.49	0.54	0.65	0.98	0.14	0.16	1.64	1.15	2.35

(1) REFLECT by Interpreted LISP on CDC6600 including garbage collection

(2) Chang & Wee by Franz LISP on VAX 11/750 including garbage collection

(3) KBROPS by OPS83 and C on PC/AT

Now we will take another example: a direct comparison between REFLECT [5], Chang & Wee's system [6] and KBROPS, to solve the same typical nine planning problems shown in Fig 10. All of them are provided in REFLECT and Chang & Wee's system. REFLECT greatly reduces the combinatorial problem by performing a preprocess to deduce the unattainable sets of state and to build macro operators. Chang & Wee's system is characterized by its goal analysis and hierarchical operator representation. The planning results are shown in the Table 2. A conclusion is that the KBROPS system has displayed more advantages.

5.2 Solution for cooperative robot planning problems

There is no similar cooperative robot problem solver to make a comparison with KBCOPS. To further illustrate the cooperative solving abilities of KBCOPS based on FM_S-1, let us choose a special example shown in Fig 11. In fact, such a problem consists of two moving sub-tasks: moving blocks A, B, and C to the goal position 1; and moving blocks D, E, and F to the goal position 2. There are only the basic cooperative requirements and communication costs. For example, if two persons were completing the task, we might let one person move blocks A, B, and C; the other move blocks D, E, and F.

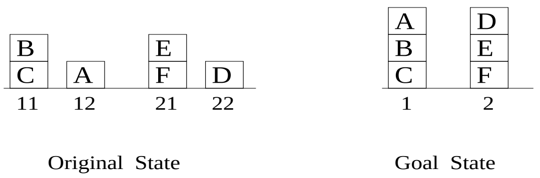


Figure 11: The special cooperative problem example

Table 3 gives four results according to four kinds of solving techniques to complete the two sub-tasks: the first, for a single robot, is implemented by OPS5; the second, for dual robots, by OPS5 (series in logical); the third for dual robots by FM_S-1 (parallel in logical, series in physical); the fourth for dual robots by FM_S-1 (parallel in physical). All the results are made in the OPS5 or OPS5-based FM_S-1 environments, and general personal computer(s).

Table 3 Comparison of solving performance

	OPS5 (single robot)	OPS5 (dual robots)	FM_S-1 (concurrent)	FM_S-1 (parallel)
Steps	28	14	14	14
Time	121.99	306.81	131.93	70.24

From the Table 3, for OPS5 (single robot), the solving time is shorter (only by 121.99s), but there are more solving steps (28 steps). For OPS5 (dual robots), the time is the longest (306.81s) as it uses a huge amount of solution space to search for the cooperative requirements, but the number of steps is only half of that using a single robot (14 steps) and the second technique begins to show enhanced solving abilities though cooperation. In the third technique, not only is the step also 14 steps, but the time is slower than in the first technique (131.93s, i.e., it only increases the cost of communication and cooperation between the dual robots, compared to the solving time of single robot). It has developed concurrent cooperative dual robot problem solving and shown a good solving performance.

Based on the parallel FM_S-1 environment of dual personal computers and only using the series communication port to meet the basic cooperation requirement, KBCOPS can show some outstanding advantages: the number of steps is also 14 steps; the time is only 70.24s (corresponding to the solving time of a single robot finishing single sub-task plus the communication time of cooperation between the two robots). In fact, it lowers the complexity of cooperative dual robots problems.

The more complex the problems, the more communication and cooperation will take place. For example, the phenomenon of enabling-condition interactions [11] has occurred. It has side-effect and if a problem contains more than one enabling-condition interaction, then it can be difficult to determine which set of actions will produce the best plan. Thus, it is necessary to use the communication and cooperation mechanism of FM_S-1 and some problem solving knowledge.

6 Conclusion

This paper establishes a direct corresponding relation between planning and combinatorial search. Thus, for one thing, all the current techniques in combinatorial search can be used in the study of planning; for another, the results in planning can promote the study of combinatorial search. In addition, the study of cooperative robot problem solving can be compared with the study of multi-agent systems.

7 Acknowledgements

The research described here is based on my past work in China. Some of the results have been presented in some journals or conferences. Currently, I am a visiting scholar in the Department of Artificial Intelligence at the University of Edinburgh, with the support of the Royal Society Exchange Fellowship. In addition, I gratefully acknowledge the computing facilities provided by the Department, and thank my host here, Dr John Hallam. Also, I appreciate all the help offered by everyone inside and outside the Department during the past twelve months of my stay.

8 References

[1] R.E.Fikes and N.J.Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, Vol.2, 1971, pp189-208.

[2] E.D.Sacerdoti, "Planning in a Hierarchy of Abstraction Space", Artificial Intelligence, Vol.5, 1974, pp115-135.

[3] S.Tangwongsan and K.-S.Fu, "An Application of Learning to Robot Planning", Int. J. of Computer and Info. Sci., Vol.4, 1979, pp303-333.

[4] Z.Cai and K.-S.Fu, "Robot Planning Expert System", Proc. of IEEE Int. Conf. on Robotics and Automation, 1986, pp1973-1978.

[5] C.Dawson and L.Silklossy, "The Role of Preprocessing in Problem Solving Systems", Proc. of IJCAI-5, 1977, pp465-471.

[6] K.-H. Chang and W.G.Wee, "A Planning System with Analysis", Proc. of 2nd Conf. on the Engineering of Knowledge-Based Systems, 1985, pp275-280.

[7] Special Volume on Planning and Scheduling, Artificial Intelligence, Vol.76, No.1-2, 1995.

[8] D.McDermott and J.Hendler, "Planning: What it is, What it could be, An Introduction to the Special Issue on Planning and Scheduling", Artificial Intelligence, Vol.76, 1995, pp1-16.

[9] B.Nebel and C.Backstrom, "On the Computational Complexity of Temporal Projection, Planning, and Plan Validation", Artificial Intelligence, Vol.66, 1994, pp125-160.

[10] M.R.Garey and D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H.Freeman and Company, 1979.

[11] N.Gupta and D.S.Nau, "On the Complexity of Blocks-World Planning", Artificial Intelligence, Vol.56, 1992, pp223-254.

[12] M.Li, et al., "Knowledge-Based Robot Planning Systems", Proc. of Int. Conf. on Information Engineering, 1991, pp414-421.

[13] R.M.Karp, "Reducibility Among Combinatorial Problems", in R.E.Miller and J.W.Thatcher (eds), Complexity of Computer Computations, Plenum Press, 1972, pp85-103.

[14] S.Sen, et al., "Learning to Coordinate without Sharing Information", Proc. of AAAI-94, pp426-431.

[15] T.Shibata and T.Fukuda, "Coordinative Behavior by Genetic Algorithm and Fuzzy in Evolutionary Multi-Agent System", Proc. of 1993 IEEE Conf. on Robotics and Automation, pp760-765.

[16] T.Ishida, "Parallel, Distributed and Multi-Agent Production Systems", Proc. of 1st Int. Conf. on Multi-Agent Systems, 1995, pp416-422.

[17] M.Li, "A General AI Problem Solving Mechanism", Proc. of 11th Int. Conf. on System Engineering, 1996, pp975-980.