2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Rearrangement task realization by multiple mobile robots with efficient calculation of task constraints

Norisuke Fujii, Tsai-Lin Chou and Jun Ota
Department of Precision Engineering
The University of Tokyo, Japan
norisuke@robot.t.u-tokyo.ac.jp
chou@robot.t.u-tokyo.ac.jp
ota@robot.t.u-tokyo.ac.jp

***Abstract*— We address a rearrangement task by multiple robot in this paper. A rearrangement task has constraints regarding the order of the start, grasping and finish time. Calculating these constraints has a high computational cost. We propose a rearrangement method that calculates constraints efficiently. In our approach, not all constraints are calculated, but some of them will be calculated step by step. The proposed method is tested in a simulated environment with up to 4 mobile robots. The methods are compared, and the results indicate that the proposed method is superior.**

I. Introduction

REARRANGEMENT tasks involving several objects are fundamental for mobile robots. Such tasks have various applications such as hazardous waste cleanup, production systems and household maintenance. The performance of such tasks by multiple, rather than single, mobile robots improves reliability, expandability and flexibility.

Figure 1 is an example of a rearrangement task. If a *task* is defined as the transportation of an object, a rearrangement task can be represented as a combination of multiple tasks. In order to start moving and transporting earlier, it is important and fundamental requirement that each robot decide which task each robot should execute and how to do it individually.

A rearrangement task contains constraints on the order of task execution. Furthermore, the order of the starting, grasping and finishing time for a task must be considered in cases involving multiple robots. In this paper, these concerns are called *task constraints*. For example, in Fig. 1, robots must grasp object 5 before the finish time of task 2 (i.e., the transportation of object 2). Considering these task constraints makes it possible to shorten the task completion time, but robots must be equipped to calculate these constraints because they cannot be pre-programmed by the designer. Furthermore, it takes a considerable amount of time to calculate constraints because consideration of the path of the robot is required.

Many researchers have studied rearrangement problems. Most of these studies involve a single robot [1], [2]. Several studies have involved multiple robots, but not all task constraints are managed, and arbitrary rules are used to avoid task constraints [3], [4]. The applied rules are only suitable when the environment does not contain task constraints. Otherwise, the time required to complete the task is considerably lengthened.

Rearrangement tasks involving multiple robots are treated as task allocation problems for multi-robot systems. In this discipline, many studies have been conducted [5], [6]. In others, such as task scheduling, constraints among tasks are also examined [7], [8]. In these studies, designers assign task constraints

All the previous methods are applicable to rearrangement problems in wide working environments when objects are simply arranged so that there are no task constraints. Otherwise, the direct application of these methods results in an increased task completion time, and, generally, the robots are prevented from performing all of the tasks.

This paper is original in that the rearrangement task has many task constraints and is performed by multiple robots. A major challenge is to shorten the task completion time. In our proposed method, robots calculate a part of, not all, the task constraints step by step. Naturally, the total path length can increase, and more time is required for execution. On the other hand, inactive times of robots are significantly reduced, and the total time for task completion is also reduced.

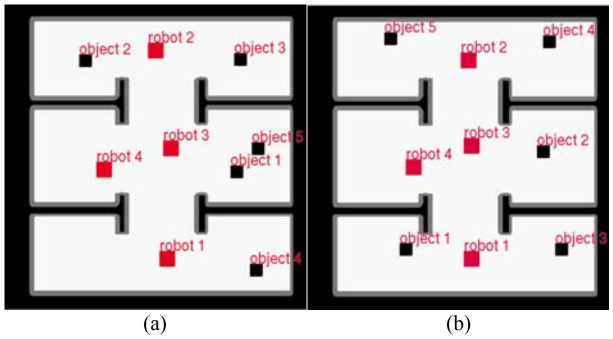


Fig. 1. A case including task constraints. (a) Initial state. (b) Goal state.

II. PROBLEM FORMULATION AND ASSUMPTIONS

A. Problem Formulation

Alami *et al.* formulated a rearrangement problem for one robot. In their research, they defined some paths in the configuration spaces of robots and objects [9]. We expand these definitions and formulation.

Let $B = \{W, R_1, \cdots, R_m, M_1, \cdots, M_n\}$ represent a set of units within the working environment. W represents immovable obstacles (i.e., walls). $\{R_1, \cdots, R_m\}$ is a collection of robots, where m is the number of robots; $\{M_1, \cdots, M_n\}$ is a collection of movable objects, where n is the number of objects. Every object and robot has its own configuration space. Let CS_{Ri} denote the configuration space of robot R_i, and let CS_{Mi} denote the configuration space of object M_i. Let CCS denote the composite configuration space of objects and robots. The set of free configurations of CCS is denoted by $Free(CCS)$. Each vector in $Free(CCS)$ is a composite configuration $Q = \{q_{R1}, \cdots, q_{Rm}, q_{M1}, \cdots, q_{Mn}\}$ in which q_{Ri} denotes the configurations of robot R_i and q_{Mi} denotes the configuration of object M_i.

The robot can only transport an object. A robot can transport an object only if it can grasp it. Grasping can be performed only when robots move to predefined configurations toward an object.

Definition 1: Such predefined configurations are called *grasping configurations*, and the grasping configuration for object M_i is represented by $Grasp(q_{Mi})$.

Definition 2: The *transfer path* of robot R_i is a path in configuration space CS_{Ri} such that there is object M_i verifying that, for any q_{Ri} on the path, $q_{Ri} = Grasp(q_{Mi})$.

Definition 3: The *transit path* of robot R_i is a path in configuration space CS_{Ri} such that there is no object verifying that, for any q_{Ri} on the path, $q_{Ri} = Grasp(q_{Mi})$.

Definition 4: The *manipulation path* of robot R_i is a path in configuration space CS_{Ri}. This path is a finite sequence of transit and transfer paths. Let $P_{Ri}(Q^0, Q^1)$ denote the manipulation path of robot R_i between Q^0 and Q^1.

Using most of the above definitions and symbols, we define the rearrangement problem of n objects and m robots as follows:

Given a description of working environment B, an initial composite configuration $Q^S = \{q^S_{R1}, \cdots, q^S_{Rm}, q^S_{M1}, \cdots q^S_{Mn}\}$ and a goal configuration $Q^G = \{q^G_{R1}, \cdots, q^G_{Rm}, q^G_{M1}, \cdots q^G_{Mn}\}$, find the all manipulation paths $\{P_{R1}(Q^S, Q^G), \cdots, P_{Rm}(Q^S, Q^G)\}$.

B. Definition of Task

The definition of the *task* is provided below:

Given the current configuration q_{Mi} and goal configuration q^G_{Mi} of object M_i, find the manipulation path of one robot to transport M_i from q_{Mi} to q^G_{Mi}.

Using the definition above, a rearrangement task can be regarded as a combination of *tasks*. At that time, task constraints and tasks that other robots engage in should be considered.

Let T_{Mi} denote the task in which a robot transports object M_i and $S = \{T_{M1}, \cdots, T_{Mn}\}$ denote the set of all tasks. Let T_{Ri} denote the task that robot R_i is now engaged in and $S_R = \{T_{R1}, \cdots, T_{Rn}\}$ denote the set of all tasks that robots are now engaged in.

C. Assumptions

In this paper, the following assumptions are made.

- There is only one grasp configuration toward one object.
- All robots can move in any direction, and the orientations of objects and robots are not considered.
- All robots are equipped with the same geometry and ability for movement, grasping and communication.
- All robots are equipped to locate objects and other robots.
- Communication among robots can be conducted whenever necessary.
- One object is transported by a single robot, and one robot can grasp one object at a time.
- The robot that is unengaged in any task tries to return to the initial position.
- The rearrangement task is completed when all tasks are accomplished and all robots return to their initial positions.

These assumptions simplify a rearrangement problem. Knowledge of other research areas can help examine the more complex versions of a rearrangement problem. In this paper, the focus is primarily on the calculation of task constraints and their influence on the outcome.

III. REARRANGEMENT METHOD

A. Overview

At the beginning of this subsection, we provide an overview of a one robot process. Figure 2 is an overview of the process. In this figure, the blue regions indicate that robots stop in these phases. The pink ones show that robots are moving. A robot iterates the selection of an object to be transported and the execution of transportation until all tasks are completed. To reflect task constraints in the selection of an object, a robot must calculate them before making a selection. However, it takes a considerable amount of time to calculate all task constraints at that time. Therefore, in this paper, robots calculate only *easy* constraints that can be calculated by a test of interference between polygons and require less computational time at that time. *Difficult* constraints that require path planning and a considerable amount of computational time are calculated when a robot cannot find a manipulation path for a selected task (i.e., when it fails). In the next subsection, we discuss which constraints are easy, which are difficult and which fall under other classifications. When a robot fails a task, there are constraints between failed tasks and others. Therefore, robots can calculate the task constraints efficiently by focusing on a specific area.

Next, we explain some specific points regarding our method.

1. Robots select one task to be executed at a time.
2. When a robot completes or fails a task, all robots reselect an object to be transported.

Regarding point 1, in a rearrangement task, robots cannot estimate the movement of other robots correctly. To avoid collisions, robots must change their paths as warranted; therefore, it is difficult to estimate the completion time of a task. According to the classification of an allocation problem by Gerkey, such tasks should be allocated one by one [10]. This is so because less information is available about future tasks and whether task constraints are satisfied.

Regarding point 2, the task constraints and priorities of all tasks can change when a robot completes or fails a task. Therefore, all robots must determine which tasks are to be executed. The robot that completes or fails a task sends a message to the other robots, and the robots that receive a message must reselect a task.

B. *Calculating Task Constraints*

In our method, we treat with just a part of task constraints, not all constraints. Task constraints can be classified primarily into three groups, as defined below:

1. Constraints prescribed by the arrangement of an object.
2. Constraints prescribed by the motion of the robots.
3. Constraints prescribed by the combination of items 1 and 2.

For the purposes of this paper, the focus is on item 1, and other constraints will not be discussed. Furthermore, the constraints discussed here are classified as *easy* and *hard* according to their calculation cost.

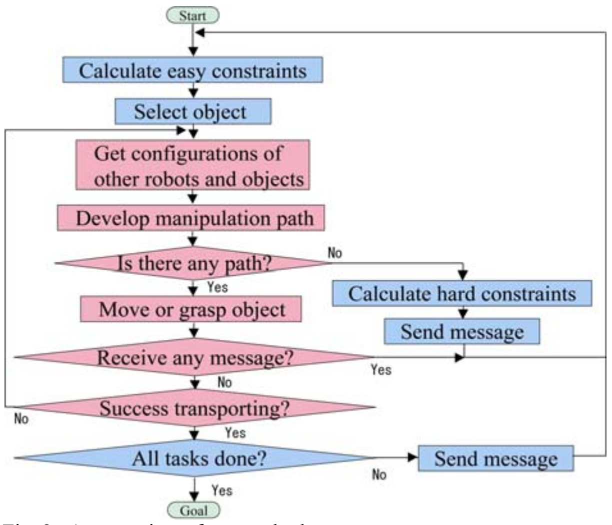


Fig. 2. An overview of our method.

1) *Easy Constraints*

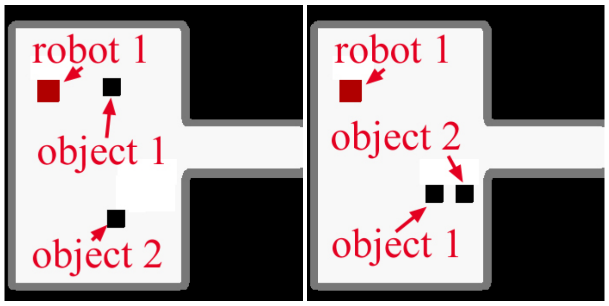
Easy constraints can be calculated by a test to determine whether two regions occupied by robot and object overlap or not. For example, the completion time of task 2 must be earlier than that of task 1. This is because, if task 1 is completed first, a robot could not be located at the grasping configuration of object 2. In a case in which initial and goal configurations are relatively near, task constraints could be calculated by testing the interference between polygons. A comparatively less computational cost is required in such a case. All easy constraints between tasks 1 and 2 could be calculated as follows:

- If object 1 at the initial configuration and a robot grasping object 2 at the initial configuration interfere, the grasping time of object 1 must be earlier than the grasping time of object 2.
- If object 1 at the goal configuration and a robot grasping object 2 at the initial configuration interfere, the grasping time of object 2 must be earlier than the completion time of object 1.
- If object 1 at the initial configuration and a robot grasping object 2 at the goal configuration interfere, the grasping time of object 1 must be earlier than the completion time of object 2.
- If object 1 at the goal configuration and a robot grasping object 2 at the goal configuration interfere, the completion time of object 2 must be earlier than that of object 1.

2) *Hard Constraints*

To calculate *hard* constraints, the iteration of path planning is required. In Fig. 4, the grasping time of task 2 must be earlier than the completion time of task 1. This kind of constraint cannot be calculated unless robots attempt to develop a path plan. Therefore, a comparatively higher computational cost is required. All hard constraints between tasks 1 and 2 can be calculated as follows:

- If there are no robots or other objects with the exception of object 1 at the initial configuration, a transfer path could be generated to transport object 2. If there is no path, the time of grasping object 1 should be earlier than the completion time of task 2.
- If there are no robots or other objects with the exception of object 1 at the goal configuration, a transfer path could be generated to transport object 2.



(a) (b)

Fig. 3. An example of easy constraints. (a) Initial state. (b) Goal state.

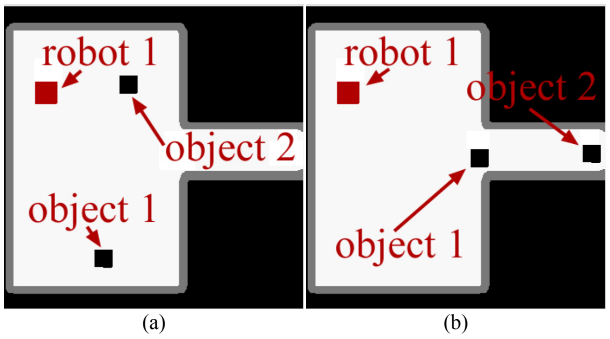


Fig. 4. An example of hard constraints. (a) Initial state. (b) Goal state.

If there is no path, the time of grasping object 2 should be earlier than the completion time of task 1.

3) Number of Constraints to be Scanned

As noted at the beginning of this section, robots calculate easy constraints before the selection of an object to be transported, and difficult ones are calculated when a robot fails a task. Regarding easy constraints, robots calculate all constraints between every two tasks. If there are n objects, the number of combinations is $n(n-1)/2$. Regarding hard constraints, robots attempt to calculate some of the constraints between a failed task and other tasks because it is more likely that there will be undiscovered constraints. The number of combinations is $n-1$.

C. Selecting the Object to be Transported

The selection of an object to be transported is carried out basically based on the priority of tasks. The more prior the task is, the sooner robots should execute that. In our method, *priority value* and *performance value* control selections. In addition to these values, we set a special selection rule in order to observe task constraints.

1) Selection Rule

The calculated task constraints must be observed in order to successfully rearrange the task. When a robot attempts to disobey a task constraint, a rearrangement task will fail. Precise estimates of the starting, grasping and finishing times are difficult to discern; therefore, a selection rule should be simple and independent of such estimates.

The selection rule is that *if there are task constraints in which the grasping or finishing times of task 1 are earlier than those of task 2, robots cannot select task 2 unless task 1 is completed.*

This rule has redundancy; therefore, our method cannot be used to obtain an optimum solution. However, this rule is sufficiently simple, and robots will always be able to observe the task constraints.

2) Priority Value and Performance Value

We use a priority graph to calculate priority values. A priority graph is a directed graph. A node denotes each task, and a directed edge from task 1 to 2 indicates that task 2 will not be selected unless task 1 is completed. The priority value of a certain node is (outdegree) + (the total priority values of nodes indicated by the edges from the node).

Figure 5 shows an example of a priority graph and values. In this example, task 3 should be executed before tasks 1 and 2, and task 2 should be executed before task 1. Eventually, a priority graph will be constructed as shown.

It is difficult to accurately calculate the time from start to finish; therefore, we approximate that time as a performance value instead. The performance values of robots are 1 / (transfer path length) supposing that there are no other objects or robots.

3) Procedure of Selection

1. Decide which tasks can be selected. In addition to the selection rule, the task with similar current and goal configurations does not require transportation; therefore, it cannot be selected. The tasks that have already been selected by other robot cannot be selected either.
2. Calculate the priority values and performance values.
3. If a robot has the highest performance value for the task with the highest priority value, it can be selected.
4. If none of the tasks can be selected or all robots have selected tasks, the procedure can be finished.
5. Return to step 1.

D. Executing Transportation

In the execution phase, the robots move along a transit path, grasp an object, continue along the transfer path and then release the object. Each robot treats other robots as stationary obstacles; therefore, robots should iterate the recognition of the environment, generate a path and move along the path. Robots move at a predefined speed. In this paper, we adopt path-planning method of Curt [11].

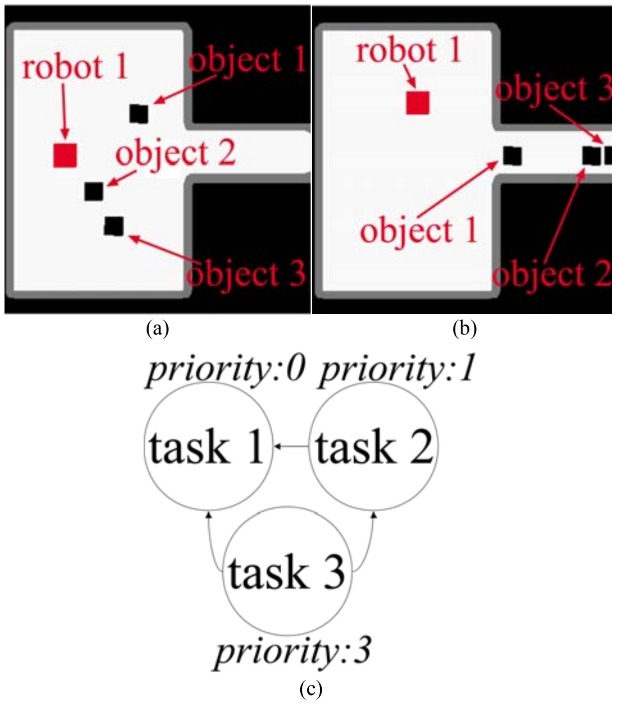


Fig. 5. An example of a priority graph and priority values. (a) Initial state. (b) Goal state. (c) Priority graph.

IV. Simulation

In this section, we present simulation results. Our method is original from the viewpoint of applicability to a narrow working environment that contains many task constraints. The application of the previous method increases the task completion time in all cases, and, in most cases, robots cannot perform all tasks.

A. Simulation Conditions

The proposed method and three other methods were tested and compared. The tested methods are summarized below:

- *Grasping-and-restricted* (proposed method): Calculate all easy task constraints just before the selection of an object and calculate some of the hard task constraints after a robot fails to execute a task.
- *All*: Calculate all easy and hard task constraints just before selection.
- *Restricted*: Calculate some of the hard task constraints after a robot fails to execute a task.
- *No-action*: Do not calculate any task constraints at all.

Next, we show other simulation conditions in Table I. All simulations were conducted with robots of the same size, speed, identically sized object and cells (these items are written in black). The sizes of the working environments and number of robots (items in red) were changed in all simulations.

B. Results

Figure 6 shows the results. The blue bars show the execution time. This time can be calculated from the path length and the speed of the robots. The purple bars show the planning time. This time is equal to the time robots remain stationary. This figure shows the effectiveness of the proposed method. The proposed method shows the same level of execution time and the best result of the planning time. The total time of task completion for the proposed method is also the best. These results show that robots can start moving faster and perform all tasks before they can with the other methods. The no-action method produces the worst results. These results show that considering task constraints is important for planning and execution. The All method produces worse results than the proposed and restricted methods. These results show that the All method wastes planning time for the calculation of task constraints and that it does not always execute well. The robot does not need to calculate all task constraints in a rearrangement problem, and it is important to determine the constraints that should be scanned. In a comparison of the proposed and restricted methods, the proposed method produced positive results. The proposed method requires much more time to calculate easy constraints than the restricted method, but it may result in good object selection. Therefore, the total completion time is minimal.

Figure 7 shows the relationship between the planning time and the number of robots. The execution times were similar among the proposed, All and restricted methods; therefore, we focused on the planning time in this figure. In our simulator, one processor always considers multiple processes of multiple robots simultaneously; therefore, when more robots are considered, more planning time is required. The results show that the more robots that are considered, the more differences there will be between the All and proposed methods. These results show the effectiveness of the proposed method, particularly in a case in which many robots are working.

Finally, a simulation in which the size of the working environment is 68.8 (m^2) with four robots working and using the grasping-and-restricted method is shown in Fig. 8.

TABLE I
SIMULATION CONDITIONS

Size of robots	0.5 m × 0.5 m
Speed of robots	0.5 m/s
Size of objects	0.4 m × 0.4 m
Size of working environment	17 m^2, 41.5 m^2, 68.8 m^2
Size of cells	0.01 m^2
Number of objects	5
Number of robots	1, 2, 3, 4
Total number of simulation times	96
CPU	Pentium 4, 3.2 GHz

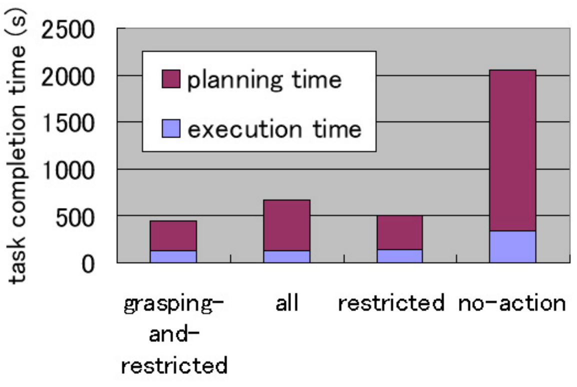


Fig. 6. Comparison of the tested methods.

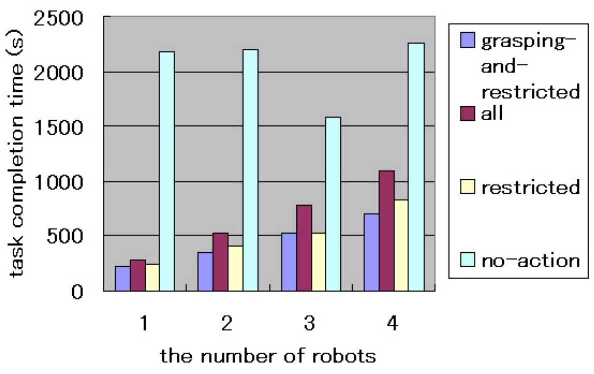


Fig. 7. Relationship between planning time and the number of robots.

V. Conclusion

In this paper, we propose a rearrangement task realization method considering multiple robots. The rearrangement task contains task constraints, and the robots should calculate these constraints. The simulation results show the effectiveness of our method from the viewpoint of the time required for task completion planning. The methods proposed in this paper do not examine all constraints. To avoid the effects from these constraints, motion planning should be improved. Future research will be conducted to further explore some of these methods.

References

[1] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Trans. Robotics and Automat.*, vol. 14, no. 4, pp. 549-565, 1998.

[2] J. Ota, "Rearrangement of multiple movable objects --- Integration of global and local planning methodology ---," in *Proc. 2004 IEEE Int. Conf. Robotics and Automat.*, pp. 1962-1967.

[3] S. Cambon, F. Grabot, and R. Alami, "aSyMov:toward more realistic robot plans," LAAS, Toulouse, Rapport LAAS N°03472 Octobre, pp. 0-8, 2003.

[4] M. Cherif and M. Vidal, "Planning handling operations in changing industrial plants," in *Proc. 1998 IEEE Int. Conf. Robotics and Automat.*, pp. 881-886.

[5] L. E. Parker, "ALLIANCE: An architecture for fault-tolerant multi-robot cooperation," *IEEE Trans. Robotics and Automat.*, vol. 14, no. 2, pp. 220-240, 1998.

[6] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multi-robot coordination," *IEEE Trans. Robotics and Automat.*, vol. 18, no. 5, pp. 758-768, 2002.

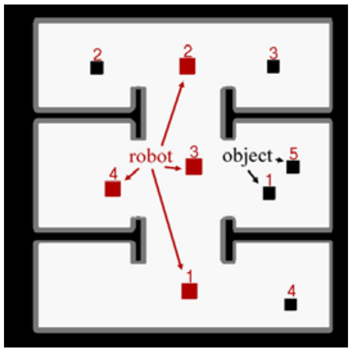
[7] J. Modi, H. Jung, M. Shen, and W. M. Kulkarni, "A dynamic distributed constraint satisfaction," in *Principles and Practice of Constraint Programming - CP2001*, T. Walsh, Ed. New York, Springer-Verlag, 2001, pp. 685-700.

[8] P. Brucker, *Scheduling Algorithms*, Berlin, Springer-Verlag, 1998.

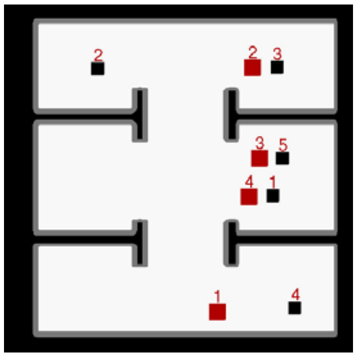
[9] R. Alami, J. P. Laumond, and T. Simeon, "Two manipulation planning algorithms," in *Algorithmic Foundation of Robotics*, K. Goldberg, D. Halpern, J. C. Latombe, and R. Wolson, Eds. Boston, A. K. Peters, 1995, pp. 109-125.

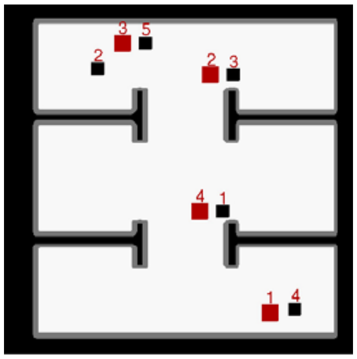
[10] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robotics Research,* vol. 23, no. 9, pp. 939-954, September, 2004.

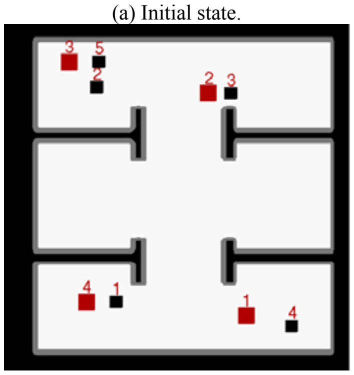
[11] K. Curt, "A gradient method for realtime robot control." In *Proc. Intelligent Robotic Systems*, vol. 1, 2000, pp. 639-646.

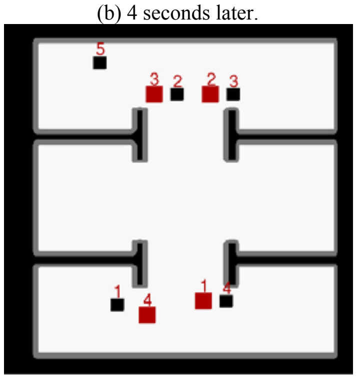


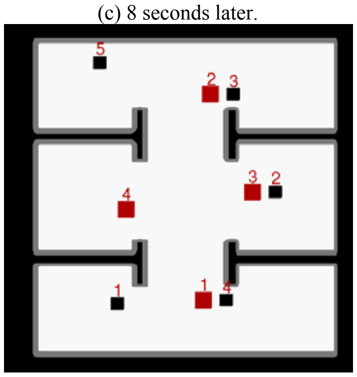
(a) Initial state.

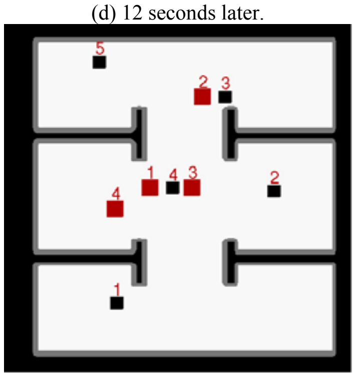

(b) 4 seconds later.

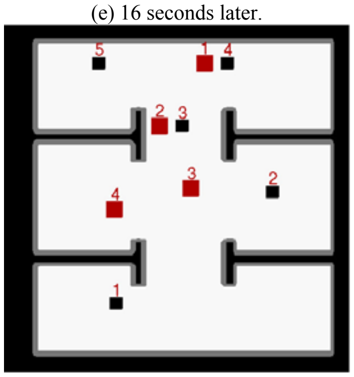

(c) 8 seconds later.


(d) 12 seconds later.


(e) 16 seconds later.


(f) 20 seconds later.


(g) 24 seconds later.


(h) 28 seconds later.

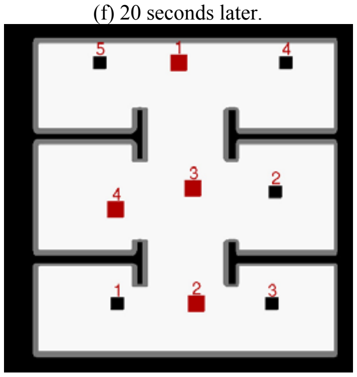

(i) Goal state.

Fig. 8. Simulation results when the size of the working environment is 68.8 (m^2) and four robots are working and using the grasping-and-restricted method.