Optimal Assignment of Robot Tasks with Precedence for Multi-Robot Coordination by Disjunctive Graphs and State-Space Search*

Chi-Fang Lin
Department of Computer Engineering and Science
Yuan-Ze Institute of Technology
Chungli, Taiwan 320
Republic of China

Wen-Hsiang Tsai†
Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 300
Republic of China

Received November 2, 1993; revised July 19, 1994;
accepted October 5, 1994

An approach to optimal assignment of tasks with precedence relationships to multiple robots is proposed. The robots are assumed to share a common workspace and work cooperatively to accomplish a given process plan consisting of a set of tasks. The optimal task assignment is defined to be the one that results in spending the least amount of time to complete the plan under the criterion that no robot collision will occur when the assigned tasks are performed. The ordering of the tasks in the process plan is described by a topological tree, which is then expanded to form a larger state-space tree without redundant tree paths. Each path in the expanded tree represents a partially developed assignment of the tasks to the robots, and a graph formulation scheme is presented for estimating the cost of the assignment. A collision-free motion schedule for each robot based on each task assignment can be obtained by finding the minimaximal path in a disjunctive graph formulated by the scheme. By using the A* algorithm, a search method for finding the optimal assignment with the minimum cost is presented. Some heuristic rules are also proposed to speed up the search process. Simulation results are illustrated to show the effectiveness of the proposed approach.

*This work was supported in part by the National Science Council, Republic of China under Contract NSC81-0404-E-009-010.
†To whom all correspondence should be addressed.

Journal of Robotic Systems 12(4), 219–236 (1995)

CCC 0741-2223/95/040219-18

複数のロボットの優先順位に関連する、タスクの最適割り当ての方法を提案する。タスクの集合で構成される処理計画を達成するために、ロボットは協調して共通の作業空間と作業を共有すると仮定している。割り当てられたタスクを実行する場合は、ロボットの衝突が起こらないことを基準にして、最短時間で計画を完了する最適タスク割り当てが定義される。処理計画におけるタスクの順番は、位相幾何学的ツリーで表現される。これは冗長なツリー経路を持たない大きな状態空間ツリーに拡張される。拡張ツリーの各経路は、ロボットのために部分的に展開されたタスクの割り当てを表している。そして、割り当ての負担を予測するためのグラフ表示について説明する。それぞれのタスク割り当てに基づいた各ロボットの無衝突動作計画は、この方法によって方程式化される分離性グラフの中からミニマックス法によって経路を探し出すことで得られる。そこで、A*アルゴリズムを使った、最少の負担で最適な割り当てを見つけだすための検索方法について説明する。また、検索処理を高速化するための、発見的規則（試行錯誤の規則）も提案する。最後に、シミュレーション結果を使って、今回提案した方法の有効性について説明する。

1. INTRODUCTION

Automatic assembly and manufacturing systems have been applied to many industrial operations to achieve high production rates. The system usually consists of a single workstation (usually including a robot) or a series of workstations (forming an assembly line) in which several robots work independently with prescribed functions such as inspection, assembly, material handling, etc. The system can be made more flexible if the robots are allowed to share a common workspace and are multifunctionally designed, given proper grippers or tools, to work cooperatively. In such a system, coordination of robot tasks is important, and given a process plan that consists of a set of tasks, it is desirable to accomplish the plan by assigning the tasks to proper robots.

As an example, Figure 1 depicts an assembly workstation with four robots and a fixture located at the center of the common workspace of the robots. Suppose that many small-sized parts in the parts feeders need be assembled according to the process plan given below:

1. load the parts from the parts feeders into the fixture;
2. drill some holes on the parts;
3. assemble the parts into two (an upper and a lower) medium-sized parts;
4. join the two medium-sized parts together to get a final product; and
5. unload the final product from the fixture to the output conveyor.

The process plan can be described by a directed acyclic graph, called a *task graph*, with nodes representing the tasks and arcs the precedence relationships, as shown in Figure 2. The process is performed repetitively unless the system is stopped or unless the process plan is changed. The problem studied here is to plan the robot motions to complete the tasks in the task graph under the criteria that the total task execution time is minimized and no collision among the robots occurs.

Previous studies on coordination of multiple robots are quite limited. Maimon[1] presented a method to implement an activity controller for a multi-robot system. The controller has the ability to coordinate any number of robots and to prevent collisions between the robots sharing a workspace. A resource supervisor was designed and assigned to each potentially shared resource, like a common workspace, to prevent unauthorized access of the resource. In Maimon,[2] a multi-level decomposition algorithm was developed to solve the robot-task-sequencing planning problem. The optimal sequence of paths for each robot to execute a prescribed task was found first, and the collision-free sequence of tasks for the

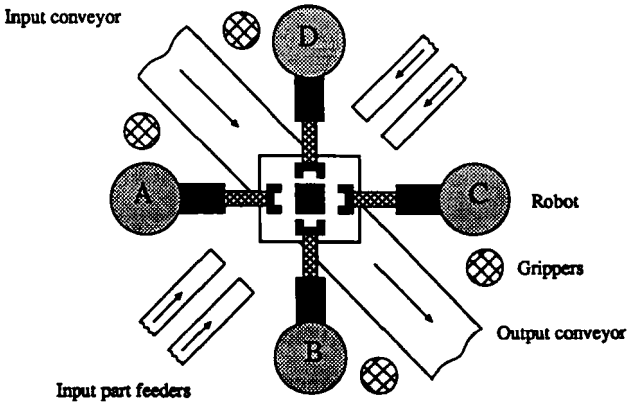


Figure 1. A multirobot system.

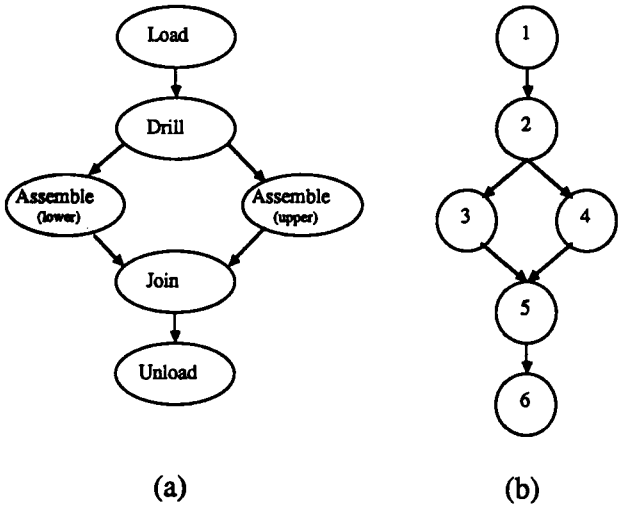


Figure 2. A process plan and a task graph: (a) the process plan; (b) the task graph of (a).

robots was determined next by formulating the problem as an integer programming problem and presenting a branch-and-bound algorithm. Nagata et al.[3] proposed a plan generation system for multiple robots. The system consists of a planning subsystem for generating sequences of actions for an assembly task, and a subsystem for detecting and avoiding mutual collisions of robots. Chen et al.[4] presented a graph-theoretic approach to determining an optimal routing assignment for a set of autonomous vehicles among several workstations. They solved the problem by a state-space search method using the A^* algorithm as well as a process of dynamic collision detection to obtain an optimal collision-free routing assignment.

In general, the above methods can only provide partial solutions for the case shown in Figure 1. To solve the problem, techniques of both motion planning for robot manipulation[5–9] and task assignment for distributed systems[10–15] are required. The former presents methods to plan collision-free motion schedules for robots performing some prescribed tasks repetitively among several moving obstacles, while the latter describes the best way of assigning tasks to processors that can maximize the throughput or minimize the total execution time. The major difference between the task assignment work for distributed systems and that for multi-robot systems is that there is no risk of physical contact among the processors when they execute tasks simultaneously; however, contact among the robots must be prevented in the latter case.

We solve the above problem by integrating techniques of both motion planning and task assignment. First, tasks in a task graph must be executed according to their precedence order, so a topological tree[5] including all the linear orderings of the tasks is first created. The tree serves as a basic structure for generating a larger state-space tree with each node indicating an assignment of a task to a robot (forming a task-robot pair). Each path of the state-space tree starting from the root node to a certain node represents a partially developed assignment consisting of several task-robot pairs corresponding to the nodes in the path. Accordingly, the cost of the assignment (or the cost of the last node in the path) is defined as the maximum completion time of the tasks assigned in the path, including not only the execution time to complete these tasks, but also the waiting time for avoiding robot collisions. This can be treated as a problem of motion planning because safe motions for multiple robots are desired. To compute the cost for each node in the tree, we present a graph formulation scheme converting the cost evaluation process into a process of path finding in a disjunctive graph. Finally, the state-space tree is searched using the A^* algorithm to find the optimal assignment with the minimum cost. Some heuristic rules are also provided for speeding up the search.

The article is organized as follows. Section 2 includes an overview of the basic concept of the proposed approach. In section 3, the motion planning method developed in a previous work[6] is reviewed, and a graph formulation scheme as well as two reduction methods are proposed. In section 4, the search method using the A^* algorithm and some heuristic rules for searching the optimal assignment are presented. Simulation results are given in section 5. Discussions and conclusions are presented, finally, in section 6.

2. PRINCIPLE OF OPTIMAL TASK ASSIGNMENT FOR MULTIPLE ROBOTS

2.1. System Assumptions and Objectives

Various assumptions made of the task graph and the robots are described in the following.

1. The robots are multifunctionally designed to perform various task by changing grippers or tools at their hands.
2. None of the robots is always given the privilege to move first when conflicts occur, i.e., the robots are all equally prioritized.

3. The task graph is a directed acyclic graph, with nodes and arcs representing the tasks and the precedence relationships between the tasks, respectively. For example, if m_1 and m_2 are two tasks and m_1 precedes m_2 in order (i.e., there exists a path from m_1 to m_2 in the task graph), then m_2 cannot be processed before m_1 is completed.
4. The total execution time for each robot to complete a given task includes the time for fetching appropriate grippers or tools, the time for executing the task, and the time for returning the grippers or tools.
5. No communication time between any two robots is considered. It is assumed that any object (e.g., an assembly part) to be processed is stationary, and placed at the center of a common workspace of the robots, so that the communication time to transmit the processing results from one robot to another is zero.

A two-dimensional function table with the value of each entry at (t, r) indicating the execution time of task t assigned to a robot r is given in advance. If task t cannot be assigned to robot r, then the value of (t, r) is set to an infinite value. An example is shown in Figure 3. Based on the above assumptions, the objective of this study now can be described as the determination of the optimal assignment of the tasks in a task graph to the robots, given the function table, under the constraints that no robot collision is allowed and that the total task execution time is minimized.

	A	B	C	D
1	5	∞	∞	∞
2	∞	6	∞	∞
3	∞	4	5	∞
4	∞	6	7	∞
5	∞	∞	5	∞
6	∞	∞	∞	5

Figure 3. An example of a function table.

More specifically, let A be a certain assignment of tasks to some robots, $t_r^e(A)$ be the time spent for task execution (including all the three types of times as stated in Assumption 4 above), and $t_r^w(A)$ the time for avoiding robot collisions (due to waiting, e.g.), both in robot r. Define $t_r(A)$ to be the total time spent in robot r for task assignment A, which is just

$$t_r(A) = t_r^e(A) + t_r^w(A). \tag{1}$$

Obviously, this amount of time is different for each distinct robot for each assignment A. Define

$$t(A) = \max_r t_r(A) \tag{2}$$

which is called the *task turnaround time* of A. It is easy to see that $t(A)$ is the total time to complete all the tasks in a task graph according to assignment A. A different assignment will result in a different amount of task turnaround time, and the smaller the task turnaround time the better the assignment. Therefore, the problem is to determine the optimal assignment A_0 that minimizes the tasks turnaround time, i.e.,

$$\begin{aligned} t(A_0) &= \min_A t(A) \\ &= \min_A \max_r t_r(A) \end{aligned} \tag{3}$$

which is the so-called *minimax criterion*.[7]

2.2. Basic Concept of Proposed Approach

Wang and Tsai[5] proposed an approach to optimal assignment of tasks with precedence relationships in distributed systems. They solved the problem by introducing the concept of a *topological tree*, which is also employed in this study. Each topological tree consists of nodes with labels corresponding to the tasks in a task graph and a dummy root node with label ϕ. Each tree path starting from the root node to a certain node represents a linear ordering (called a topological ordering[8]) of the tasks in the path. The topological tree for the task graph shown in Figure 2 is depicted in Figure 4a. The topological tree can be used as a basic structure for generating a larger state-space tree. The generation procedure is reviewed as follows, in which the term "processor" corresponds to "robot" for our case of task assignment here.

Step 1. Generate a dummy node O_e with label ϕ as the root node of the state-space tree corresponding to the root node O_t of the topological tree.
Step 2. Expand O_e as follows:

1. collect all sons of O_t into a set $\mathcal{A}$;
2. for each node in $\mathcal{A}$ with label m and each processor p, check the candidate pair (m, p) for the validity, i.e., check if m is really executable on p; and
3. expand node O_e by generating as sons all the valid task-processor pairs, each with the corresponding label (m, p).

Step 3. Expand non-root node (m, p) as follows:

1. identify the corresponding node in the topological tree with label m; and
2. similarly to the expansion of the root node discussed above, collect all sons of m and generate the appropriate valid task-processor pairs as the sons of node (m, p).

Step 4. Repeat Step 3 until all non-root nodes are expanded.

The expanded state-space tree for Figure 4a is shown in Figure 4b, given the function table of Figure 3. By computing an appropriate cost for each node in the expanded tree, and searching the tree using the well-known A^* algorithm, the optimal task assignment with the minimum cost can be found.

The concept of topological tree, as well as the tree expanding method, are used to solve the problem of task assignment for multiple robots in this study. Unlike the case studied in Wang and Tsai,[5] collisions among the robots should be considered and avoided in the multi-robot systems. A cost derivation method considering both the task execution time and the waiting time for avoiding robot collisions is proposed. The concept is based on the observation that for each path starting from the root node of the expanded tree to a certain node n, there may exist one or more robots performing the corresponding tasks simultaneously. The cost of node n (or the cost of the partially developed assignment) can be defined to be the maximum execution time for the robots to complete these tasks (or the maximum completion time of the tasks), under the criterion that no robot collisions occur. In more specific terms, assume that the partially developed assignment is denoted as A'; then the cost C_n of node n is computed by

$$C_n = \max_r t_r(A'). \tag{4}$$

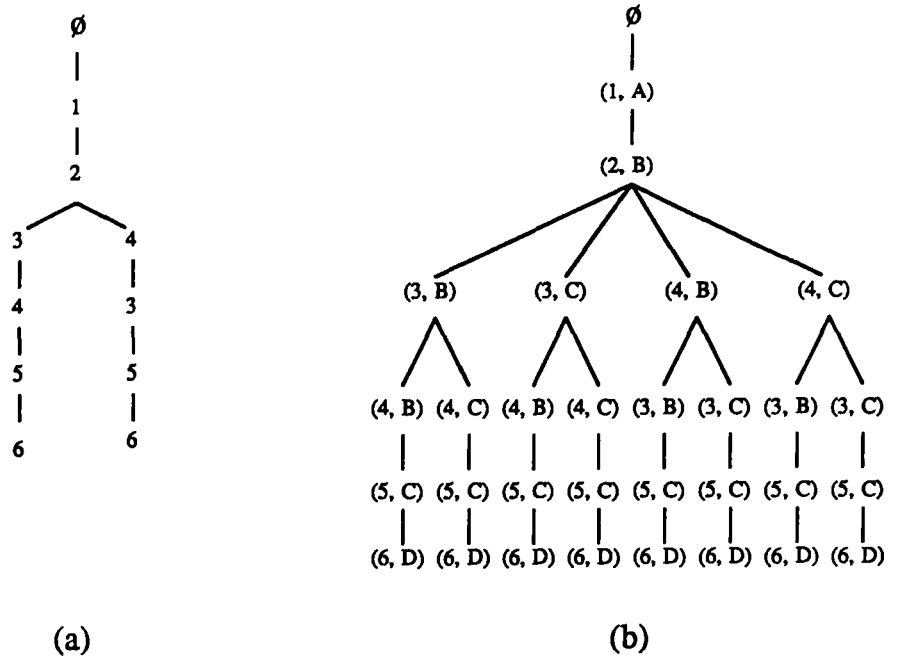


Figure 4. A topological tree and a fully expanded state-space tree: (a) the topological tree corresponding to the task graph shown in Fig. 2b; (b) the state-space tree generated by expanding (a).

Computation of each node cost is formulated as a problem of motion planning by a graph formulation scheme developed in this study, and the cost derivation process thus can be regarded as a sequence of motion planning processes, each for deriving the cost of a node. This scheme is then used in evaluating the cost value of each node n in the state-space tree of the A^* algorithm. The details will be discussed in sections 3 and 4.

3. COST DERIVATION FOR TASK ASSIGNMENT

To evaluate the cost of a certain node n in the expanded topological tree, which is defined as the cost of a partially developed assignment consisting of the task-robot pairs obtained by backtracking from node n to the root node, a graph formulation scheme is proposed to transform the cost derivation process into a process of path finding in a disjunctive graph. The motion planning method developed by Lin and Tsai[6] is then applied to the graph constructed by the scheme, and the result is described by Gantt charts, each being a collision-free motion schedule for a robot according to this assignment. The cost of node n is just the maximum length measured in time of the charts. To reduce the cost computation time for each node and the search time for finding the optimal assignment, two reduction methods are also proposed. The details will be discussed in the subsequent sections.

3.1. Review of Motion Planning via Disjunctive Graphs

The motion planning problem is solved in Lin and Tsai[6] for multiple robots by employing the concept of a disjunctive graph[9–11] as follows. First, a schedule map[12] for each pair of robots performing certain tasks is created. The map is a two-dimensional figure with the horizontal axis and the vertical one representing, respectively, the execution time flows of two robots, say R_1 and R_2, to complete the specified tasks. The task performed on each robot conceptually can be divided into a series of pseudo-subtasks, each requiring an identical amount of processing time H (the execution time of the task is assumed to be composed of a multiple of small time intervals), and these subtasks are indexed sequentially and increasingly.

For example, the subtasks are numbered from 1 through n_1 for R_1 and from $n_1 + 1$ through $n_1 + n_2$ for R_2, if the number of subtasks performed on R_1 and R_2 are n_1 and n_2, respectively. A simple example is shown in Figure 5a. Also shown in this figure are some non-blank (shaded or dark) squares called collision squares,[6] indicating that a collision will occur if both robots perform the corresponding subtasks simultaneously. Each set of adjacent collision squares are grouped together to form a collision region. A pair of collision squares, one on the upper left position and the other on the lower right of a rectangle satisfying the following two conditions, are termed guiding squares[6]: (1) they are in the same collision region; and (2) the rectangle specified by the two collision squares is maximal in the sense that it cannot be contained within any other rectangle defined by a different pair of collision squares in the region. The guiding squares are shown dark in Figure 5a, and they will be used to define the disjunctive arcs for a disjunctive graph in the next section.

In addition, the notation $MAP_{(t_m,t_n;R_r,R_s)}$ is used in this study to denote the schedule map of robots R_r and R_s performing tasks t_m and t_n, respectively. It is noted that $MAP_{(t_m,t_n;R_r,R_s)}$ need be created only when both tasks t_m and t_n are unrelated in the task graph (i.e., only when there exists no path from t_m to t_n and no path from t_n to t_m in the graph). The reason is that, if the two tasks are related so that if a task, say t_m, is the predecessor of the other, then t_m must be completed before t_n can be started. Therefore, no conflict will occur because t_m and t_n cannot be processed simultaneously at any time instant.

After the creation of the schedule maps as discussed above, a disjunctive graph is defined as $G = (\mathcal{V}; \mathcal{C}, \mathcal{D})$, where (1) $\mathcal{V}$ is a set of nodes with labels corresponding to the subtasks and two additional dummy nodes, labeled 0 and *, indicating the source and the sink of the graph, respectively; (2) $\mathcal{C}$ is a set of conjunctive arcs with each element $(i, i + 1)$ representing that subtasks i and $i + 1$ are performed by an identical robot and subtask i precedes subtask $i + 1$ in order; $\mathcal{C}$ also includes, for each robot R_k, two additional arcs $(0, s_k)$ and $(e_k, *)$ where s_k and e_k are the first and the last subtasks performed on robot R_k, respectively; (3) $\mathcal{D}$ is a set of disjunctive arc pairs with each pair of arcs (i, j) and (j, i) indicating that subtasks i and j are performed on different robots but their corresponding time intervals span a guiding square. The disjunctive graph of Figure 5a is shown in Figure 5b. The amount of processing time (or the arc length) associated with each arc (i, j) in $\mathcal{C} \cup \mathcal{D}$ is

$$\begin{cases} H & \text{if } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

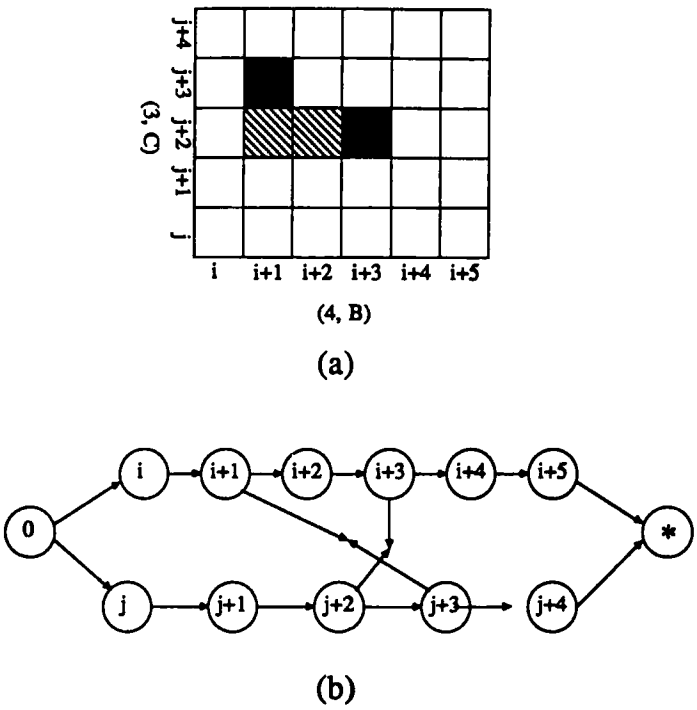


Figure 5. A schedule map and its corresponding disjunctive graph: (a) the schedule map of robots B and C performing tasks 4 and 3, respectively; (b) the disjunctive graph representation of (a).

where H is a positive constant specifying the amount of processing time of each subtask. By finding the minimaximal path in a disjunctive graph, the problem of motion planning for multiple robots can be solved for various conditions.[6]

3.2. Disjunctive Graph Formulation Scheme

Consider that a path in the expanded tree with nodes ϕ, (t_1, R_1), . . . , (t_n, R_n), where ϕ is the root node of the tree and node (t_i, R_i) represents the assignment of task t_i to robot R_i. The cost C_n of node (t_n, R_n) has been defined to be the maximum completion time of all tasks t_i, for $i = 1, 2, \ldots, n$, as described by Eq. (4). Because there may exist more than one robot in the path performing the corresponding assigned tasks simultaneously, additional costs (e.g., the time for waiting) for avoiding robot collisions should be considered. This problem can be treated as a problem of motion planning.

Assume that $MAP_{(t_m,t_n;R_r,R_s)}$ is the schedule map as defined previously, and let $G = (\mathcal{V}; \mathcal{C}, \mathcal{D})$ be a disjunctive graph created as discussed in the last section. Define the outdegree of node u in $\mathcal{V}$ to be the number of arcs originating from u and pointing outward, and the indegree of node u the number of arcs originating from the other nodes in $\mathcal{V}$ and pointing to u. Before the graph formulation scheme is presented, several operators applicable to G are introduced and listed in the following (assume that $1 \leq i, j \leq n$).

1. *The assignment operator*: Applying the assignment operator to node (t_i, R_i) means adding into $\mathcal{V}$ the nodes with labels corresponding to the subtasks of task t_i, and adding into $\mathcal{C}$ a set of conjunctive arcs $(i, i + 1)$ in which i and $i + 1$ are subtasks performed on robot R_i (see Fig. 6a, given the task graph and the function table shown in Fig. 2b and Fig. 3, respectively).
2. *The conjunction operator*: Applying the conjunction operator to a pair of nodes $((t_i, R_i), (t_j, R_j))$ means adding into $\mathcal{C}$ a conjunctive arc (e_i, s_j), where e_i and s_j are the last and the first subtasks of t_i and t_j, respectively (i.e., the last and the first subtasks performed on R_i and R_j, respectively) (see Fig. 6b). The conjunction operator is applied when t_i is a predecessor of t_j or when both R_i and R_j denote the same robot.

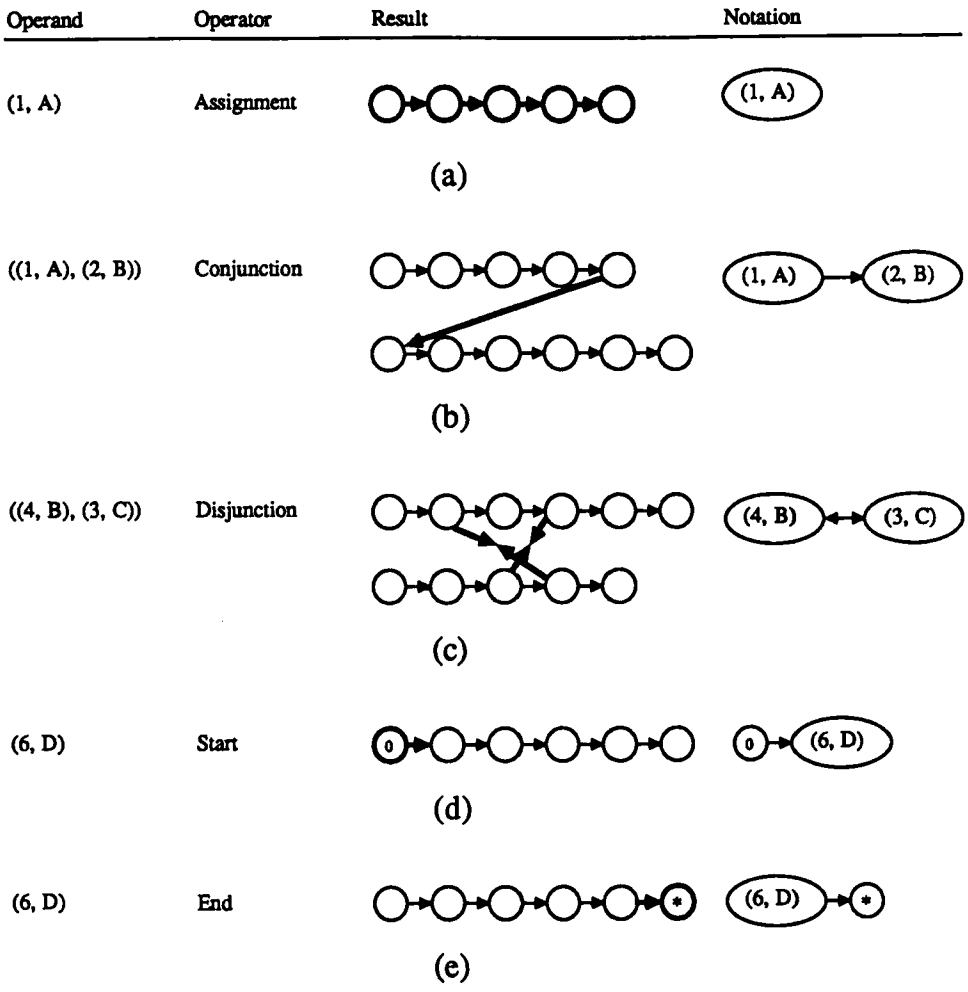


Figure 6. Operators for creating a disjunctive graph (newly-created arcs and nodes generated by the corresponding operators are shown dark): (a) the assignment operator; (b) the conjunction operator; (c) the disjunction operator; (d) the start operator; (e) the end operator.

3. *The disjunction operator*: Applying the disjunction operator to a pair of nodes $((t_i, R_i), (t_j, R_j))$ means adding into $\mathcal{D}$ pairs of disjunctive arcs, with each pair of arcs (u_i, v_j) and (v_j, u_i) specifying the guiding square in $MAP_{(t_i,t_j;R_i,R_j)}$ spanned by the time intervals of u_i and v_j, where u_i and v_j are the subtasks of t_i and t_j, respectively (see Fig. 6c). The disjunction operator is applied when t_i and t_j are unrelated, which means that there exists no path from t_i to t_j and no path from t_j to t_i in the task graph.
4. *The start operator*: Applying the start operator to a set of nodes $((t_i, R_i), (t_{i+1}, R_{i+1}), \ldots, (t_j, R_j))$ means adding a dummy node 0 into $\mathcal{V}$ and a set of conjunctive arcs $(0, u)$ into $\mathcal{C}$, where u is a node in $\mathcal{V} - \{0\}$ and the indegree of u is zero (see Fig. 6d).
5. *The end operator*: Applying the end operator to a set of nodes $((t_i, R_i), (t_{i+1}, R_{i+1}), \ldots, (t_j, R_j))$ means adding a dummy node $*$ into $\mathcal{V}$ and a set of conjunctive arcs $(v, *)$ into $\mathcal{C}$, where v is a node in $\mathcal{V} - \{*\}$ and the outdegree of v is zero (see Fig. 6e).

The algorithm to create a disjunctive graph for computing the cost of node (t_n, R_n) is now presented as follows.

Algorithm 1. Disjunctive graph creation algorithm.

Input: The task graph and the tree path with nodes ϕ, (t_1, R_1), . . . , (t_n, R_n) in the expanded topological tree.

Output: A disjunctive graph $G = (\mathcal{V}; \mathcal{C}, \mathcal{D})$.

Method:

Step 1. Apply the assignment operator to each node (t_i, R_i) in the path for $i = 1, 2 . . . , n$.

Step 2. For each pair of nodes (t_i, R_i) and (t_j, R_j) in the path where $1 \leq i < j \leq n$, the following conditions are checked.

Condition 1. If R_i is identical to R_j (i.e., if they denote an identical robot), then apply the conjunction operator to $((t_i, R_i), (t_j, R_j))$.

Condition 2. If R_i is not identical to R_j (i.e., if they do not denote an identical robot) and t_i is the predecessor of t_j in the task graph, then apply the conjunction operator to $((t_i, R_i), (t_j, R_j))$.

Condition 3. If R_i is not identical to R_j and t_i is unrelated to t_j in the task graph, then apply the disjunction operator to $((t_i, R_i), (t_j, R_j))$.

Step 3. Apply the start and the end operators to the set of nodes $((t_1, R_1), (t_2, R_2), . . . , (t_n, R_n))$ and stop.

The graph output by Algorithm 1 can then be used as input to the motion planning algorithm,[6] and the output (i.e., the planning result) will include the maximum completion time of task t_i, for $i = 1, 2, . . . , n$, which is just the cost of node (t_n, R_n).

3.3. Reductions of Conjunctive Arcs and Tree Paths

3.3.1. Reduction of Conjunctive Arcs

The disjunctive graph created in the last section may contain certain redundant conjunctive arcs that are useless and can be depleted from $\mathcal{C}$ without changing the final scheduling result. To see an example, assume that (t_i, R_i), (t_j, R_j), and (t_k, R_k) are three different task-robot pairs corresponding to the nodes in the expanded tree. By applying the assignment operator to these nodes and the conjunction operator to pairs of nodes $((t_i, R_i), (t_j, R_j))$, $((t_j, R_j), (t_k, R_k))$, and $((t_i, R_i), (t_k, R_k))$, the resulting graph is shown in Figure 7. It can be seen from this figure that the conjunctive arc that connects the last subtask of t_i to the first subtask of t_k is redundant because it cannot

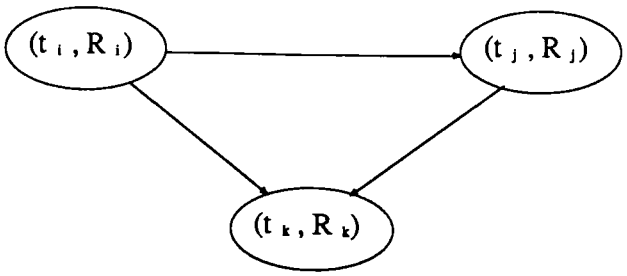


Figure 7. An example of a redundant conjunctive arc.

be the arc on any of the longest paths from the first subtask of t_i to the last subtask of t_k.

A method to identify and delete the redundant conjunctive arcs is presented here. Suppose that $A = [a_{ij}]$ is an adjacency matrix with size $|\mathcal{V}| \times |\mathcal{V}|$ for graph $G = (\mathcal{V}; \mathcal{C}, \mathcal{D})$, where $|\mathcal{V}|$ is the number of nodes in $\mathcal{V}$, and for any i, j in $\mathcal{V}$, $a_{ij} = 1$ if there is an arc (i, j) in $\mathcal{C}$ and $a_{ij} = 0$ if not. The number of paths between vertices i and j is given by the value of the ij-element in the matrix $B = [b_{ij}]$.[13] Then, it can be seen that arc (i, j) in $\mathcal{C}$ is redundant and can be deleted from $\mathcal{C}$ if

$$a_{ij} \cdot b_{ij} > 1. \tag{5}$$

The reason is that if there is more than one path between vertices i and j, then arc (i, j) must be the shortest path connecting both i and j (it is defined before that the processing time associated with each arc in $\mathcal{C} \cup \mathcal{D}$ is a constant H, and is treated as the length of the arc).

3.3.2. Reduction of Tree Paths

Sometimes disjunctive graphs created from different tree paths using Algorithm 1 may actually be identical. For example, in Figure 8a the disjunctive graph created from the left path $(\phi, (1, A), (2, B), (3, B), (4, C))$ is identical to that created from the right path $(\phi, (1, A), (2, B), (4, C), (3, B))$, and the graph is shown in Figure 8b. The tree paths generating an identical disjunctive graph are thus redundant except for one of them. Time will be wasted in computing the node costs and in searching the optimal assignment if the expanded topological tree contains too many such redundant paths. The expanding method of Wang and Tsai[5] can be used to generate expanded topological trees, but it also generates redundant paths. This problem is solved in this study by expanding the topological tree in a variable-sized form instead of in a fixed-sized form like that pro-

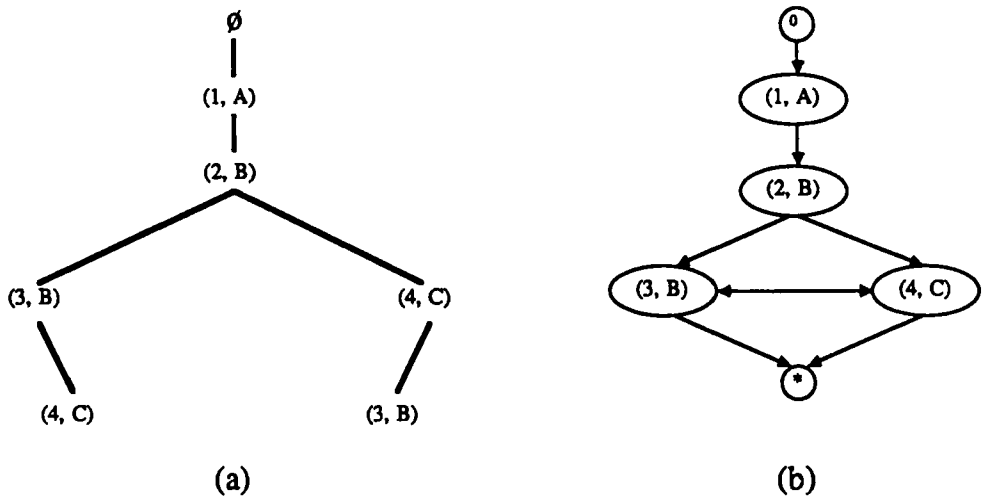


Figure 8. An example of redundant tree paths: (a) two different tree paths of Fig. 4b corresponding to an identical graph representation; (b) the identical disjunctive graph of (a).

posed in Wang and Tsai.[5] Assume that the robots are indexed with distinct numbers. The method is to modify the generation process discussed in section 2.2 for expanding a non-root node (m, R_i) as follows.

1. Identify the corresponding node in the topological tree with label m.
2. Collect all sons of m in the task graph and generate all task-robot pairs as sons of node (m, R_i), with each pair (n, R_j) satisfying the following two requirements: (i) task n can be performed on robot R_j according to the function table; and (ii) if task n is unrelated to task m in the task graph, then the index of R_j must be greater than or identical to that of R_i (otherwise redundant tree paths will be created).

The result of applying the above modified process to Figure 4a is shown in Figure 9. The effectiveness of the improved method will be illustrated by simulation experiment results given in section 5.

4. SEARCHING FOR OPTIMAL ASSIGNMENT

The expanded topological tree is useful for generating all possible assignments of the tasks in a task graph to the robots. The cost of each node can be determined by the methods proposed in the last section. Searching for the optimal assignment with the minimum cost is a state-space search problem, and the A^* algorithm discussed in Nilsson[14] is used in this study. The expanded topological tree serves as a state-space search tree here. Some heuristic rules are also provided to speed up the search process. The details are described in this section.

4.1. Search Method

In a state-space search problem, each state is described as a node, and an operator is designed to

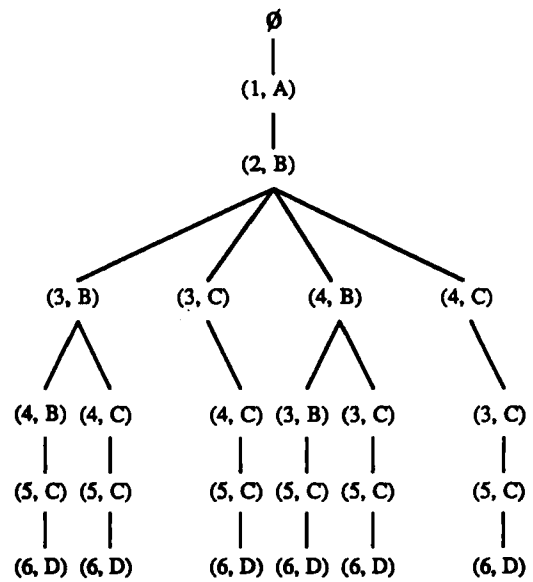


Figure 9. The tree of Fig. 4(b) containing no redundant tree path.

generate all the successors for the node, called *node expansion*. A solution path of a search problem is a path starting from the initial state to one of the goal states. In our case here, a solution path defines an assignment of the tasks in a task graph to the robots. Note that the cost of a solution path is actually the cost of a terminal node in the path. The search method is now described as follows.

State Description. Let two-tuple $S_n = (P_n, C_n)$ denote the description of the current search state at node n, where $P_n = \{(t, r)|(t, r)$ is a task-robot pair associated with a node in the path starting from the root node to node $n\}$ and C_n is the cost of node n defined in section 2. Also let $T_n = \{t|(t, r)$ is in $P_n\}$ be a set of already assigned tasks.

Initial State. The initial state is $S_\phi = (P_\phi, C_\phi)$ with P_ϕ being an empty set and C_ϕ zero, where ϕ is used to denote the root node.

Operator. The procedure of applying an operator to node n (called node expansion) with node label (t,r) and state description $S_n = (P_n, C_n)$ is described as follows. First, generate all the sons of n according to the improved expanding method discussed in section 3.4. Next, update S_n for each generated node m with node label (t', r') as $S_m = (P_m, C_m)$, where $P_m = P_n \cup (t', r')$ and C_m is the cost of node m computed using the method presented in section 3.2.

Goal State. Any state $S_g = (P_g, C_g)$ is a goal state if and only if the corresponding T_g contains all the tasks in a task graph. Node g is termed a goal node.

In the A^* algorithm, the node to be expanded in every step is determined by a cost evaluation function, and the solution path with the minimum cost, called the optimal solution path, can always be found if the cost function is properly defined. More specifically, let $f(n)$ be a cost function, and

$$f(n) = g(n) + h(n) \tag{6}$$

where $g(n)$ is a non-negative measure of the cost of the path from the root node to node n in the state-space tree, and $h(n)$ is a lower-bounded estimate of $h^*(n)$, which is the minimum cost of the path from node n to a goal node. According to Nilsson,[14] the optimal solution path is guaranteed as long as $h(n) \leq h^*(n)$ for all n (i.e., as long as $h(n)$ is *consistent*), if one exists. This is equivalent to saying that if $f(n) \leq f^*(n)$ for all node n where

$$f^*(n) = g(n) + h^*(n), \tag{7}$$

then the optimal assignment A_0 in Eq. (3) can always be found using the A^* algorithm.

Algorithm 2. Find the optimal assignment A_0.

Step 1. Put the root node ϕ into a list called OPEN, and set the value of $f(\phi)$ to zero.
Step 2. Remove from OPEN the node n with the smallest f value (to be estimated in the next section), and put the node into a list called CLOSED.
Step 3. If node n is the goal state, then backtrack from n to ϕ to obtain the corresponding assignment A_0 and exit; otherwise continue.
Step 4. Expand node n and update the state description for each generated node using the above operators. Append the generated nodes and the corresponding f values to the OPEN list.
Step 5. Go to Step 2.

4.2. Evaluation of Cost Function

To evaluate the value of function f for each node generated in Step 4 of the above algorithm, a method consisting mainly of the following two steps is proposed: (1) creating two disjunctive subgraphs G_g and G_h corresponding to the function g and the heuristic function h, respectively; and (2) combining G_g and G_h into a larger graph G_f, which is then taken as the input into an algorithm described in our previous work[6] for evaluating the cost of f. The operators listed in section 3.2 are used in constructing these subgraphs. We also provide some heuristic rules for use in constructing G_h and in combining the two subgraphs to guarantee that the consistency property of f is satisfied (i.e., to guarantee the inequality $f(n) \leq f^*(n)$ for all nodes in the state-space tree).

Assume that $f(n)$ of node n with state description $S_n = (P_n, C_n)$ and set T_n as defined in section 4.1 is to be estimated. Let $\overline{T}_n$ be the complement of T_n (i.e., $\overline{T}_n$ is the set of unassigned tasks in the task graph and $T_n \cap \overline{T}_n = \phi$). The method for estimating the value of $f(n)$ consists of the following two steps.

Step 1. Construction of G_g and G_h.

As mentioned before, function $g(n)$ is a measure of the cost of the path from the root node to node n. Note that g is a measure of a true cost; it is not an estimated value. The cost of node n can be used as the value of $g(n)$, and Algorithm 1 discussed in section 3.2, excluding Step 3, can be applied to construct the subgraph G_g.

On the other hand, function $h(n)$ is a lower-

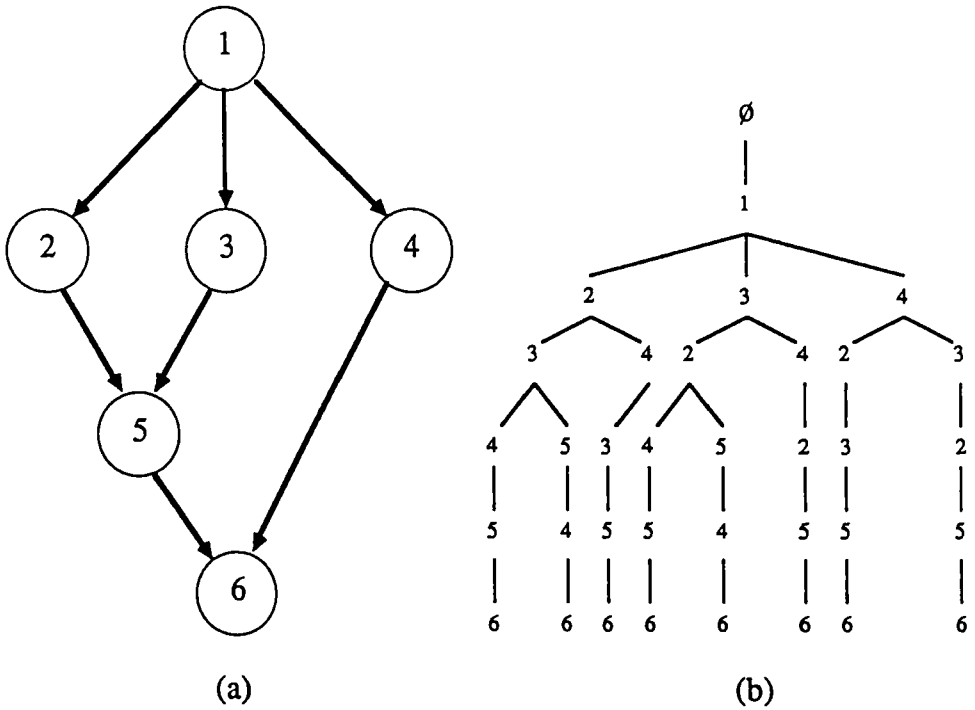


Figure 10. A task graph and the corresponding topological tree: (a) the task graph; (b) the tree including all the topological orderings of the tasks in (a).

bounded estimate of the minimum cost h^* of the path from node n to a goal node. We provide two heuristic rules for creating the subgraph G_h. The first rule utilizes the set of unassigned tasks $\overline{T}_n$. It is easy to know that each task in $\overline{T}_n$ must be assigned to one and exactly one robot to complete the process plan. To ensure that the estimated value h is never larger than h^*, each of the remaining unassigned tasks can be assigned to the robot that spends the least amount of time to complete the task. In more specific terms, we assign each task t in $\overline{T}_n$ to a certain robot r which is determined by

$$r = \min_{r'} t^e(t, r') \tag{8}$$

where $t^e(t, r')$ is the execution time of robot r' to complete task t. Assume that the label of node n considered here is (t_n, R_n). Note that if task t in Eq. (8) is unrelated to task t_n of node n, an additional constraint to be checked is that the index of robot r determined by the above equation must be greater than or identical to that of robot R_n. The reason is that the state-space tree generated in this study is not a fully expanded tree because the redundant tree paths have been removed. This fact should be considered in determining the robot for each unassigned task in Eq. (8). After finding appropriate task-robot pairs by Eq. (8), the assignment operator is applied to each of the pairs.

The second rule involves the processing sequence of the task-robot pairs assigned by Rule 1. Assume that t_i and t_j are two tasks in $\overline{T}_n$, and t_i precedes t_j in the task graph. It is easy to see that no matter to what robots the two tasks are assigned, t_i must be completed before t_j can be started. So, for any two task-robot pairs created by Rule 1 and processed by the assignment operator, say (t_i, R_i) and (t_j, R_j), if task t_i is a predecessor of task t_j, then apply the conjunction operator to $((t_i, R_i), (t_j, R_j))$, and the subgraph G_h is hence created.

Step 2. Combination of G_g and G_h.

The conjunction operator is used again in this step to combine the two subgraphs G_g and G_h created in the last step. The operator is applied to each pair of nodes $((t_i, R_i), (t_j, R_j))$, where task t_i is in T_n and task t_j is in $\overline{T}_n$, and the former precedes the latter in order. The start and the end operators are applied next to the combined result of G_g and G_h to create graph G_f. Finally, the redundant conjunctive arcs in graph G_f are identified and deleted using the method presented in section 3.3 to achieve a better performance.

The value of $f(n)$ can be obtained by taking graph G_f as input to an algorithm from our previous work.[6] Note that if the value of h is always set to zero, then the search method discussed above will be a uniform-cost search.[14] It can be seen that the consistency requirement is satisfied in evaluating f using the above methods.

5. SIMULATION RESULTS

Three illustrative examples are given in this section to show the effectiveness of the proposed approach. Four robots (A, B, C, and D) are scheduled to execute six tasks in our simulation experiments. A function table indicating the processing time of each task performed on each robot is given in Figure 11. A task graph describing the precedence relationships of the tasks and a set of schedule maps constructed in advance are also given for each example and shown in Figures 12 through 14.

By using Algorithm 2 presented in section 4 as a search method, the optimal assignment of the tasks to the robots with the minimum task turnaround time can be obtained for each simulation example. To show the effectiveness of our heuristic rules in search speed and reduction of the number of searched nodes, we further reuse Algorithm 2 as a uniform-cost search algorithm by setting $h(n)$ zero for all nodes. The results are summarized in Table I. The total number of nodes generated in the state-space tree for Example 1, as shown in Table I, if expanded in a fixed-sized form, is 959, and is reduced to 349 if expanded in a variable-sized form. Among these nodes, as shown in Figure 15a, in total only 16 nodes are expanded and 43 nodes generated before the goal node is found by using our search method, and 109 expanded and 206 generated if the uniform-cost search method is employed (the resulting graph for the latter case is too large to be included as a figure and is omitted here). The optimal assignment is (1, D), (2, A), (3, C), (4, B), (5, D), and (6, A), and the cost is $33H$. The collision-free motion schedules of the four robots are depicted by Gantt charts in Figure 15b.

	A	B	C	D
1	9	∞	∞	8
2	7	8	∞	∞
3	∞	10	7	∞
4	∞	8	6	∞
5	∞	∞	10	9
6	7	∞	∞	9

(unit : H)

Figure 11. A function table for the illustrative experiments.

Table I. A summary of the simulation results

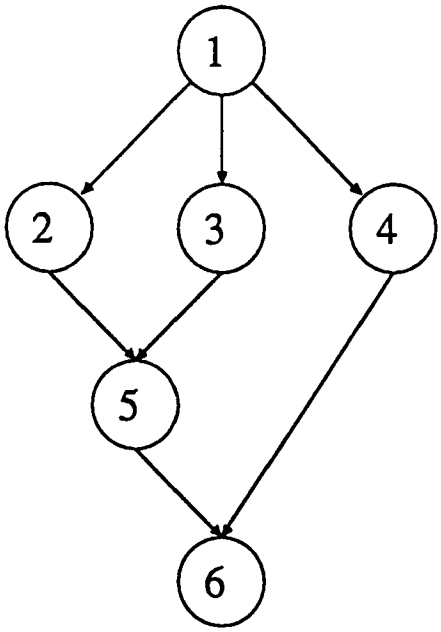
Examples	Example 1	Example 2	Example 3
# of nodes (without reduction)	959	1,707	2,511
# of nodes (with reduction)	349	509	422
Nodes expanded (A* search)	16	16	15
Nodes generated (A* search)	43	39	38
Processing time@ (A* search)	1.43	1.76	1.70
Nodes expanded (uniform-cost search)	109	61	39
Nodes generated (uniform-cost search)	206	169	95
Processing time@ (uniform-cost search)	16.70	20.27	4.07
Optimal cost	33	26	17

@ = CPU sec. on PC386.

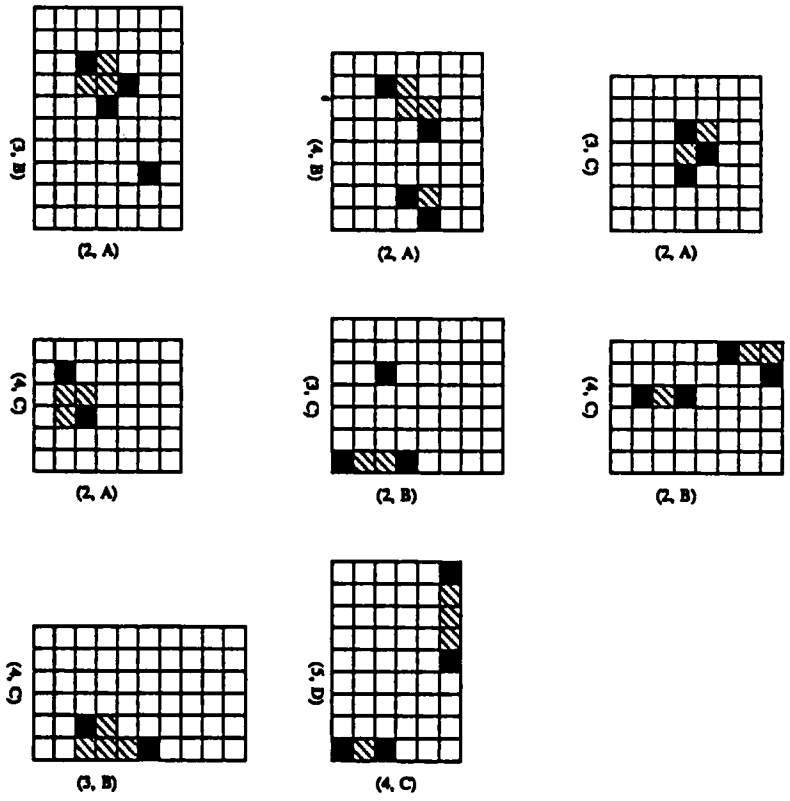
The simulation experiments were programmed using the C language on a personal computer (IBM PC/AT). The use of faster workstations may improve the speed.

6. CONCLUSIONS AND DISCUSSIONS

The problem of optimal assignment of tasks with precedence relationships to multiple robots is solved in this study. The proposed approach consists of the following three major parts: (1) a state-space tree generation method considering all possible assignments of the tasks in a task graph to the robots; (2) a cost derivation method that can evaluate the processing time for each assignment of the tasks to the robots, including the execution time to complete



(a)



(b)

Figure 12. Example 1: (a) the task graph; (b) the schedule maps.

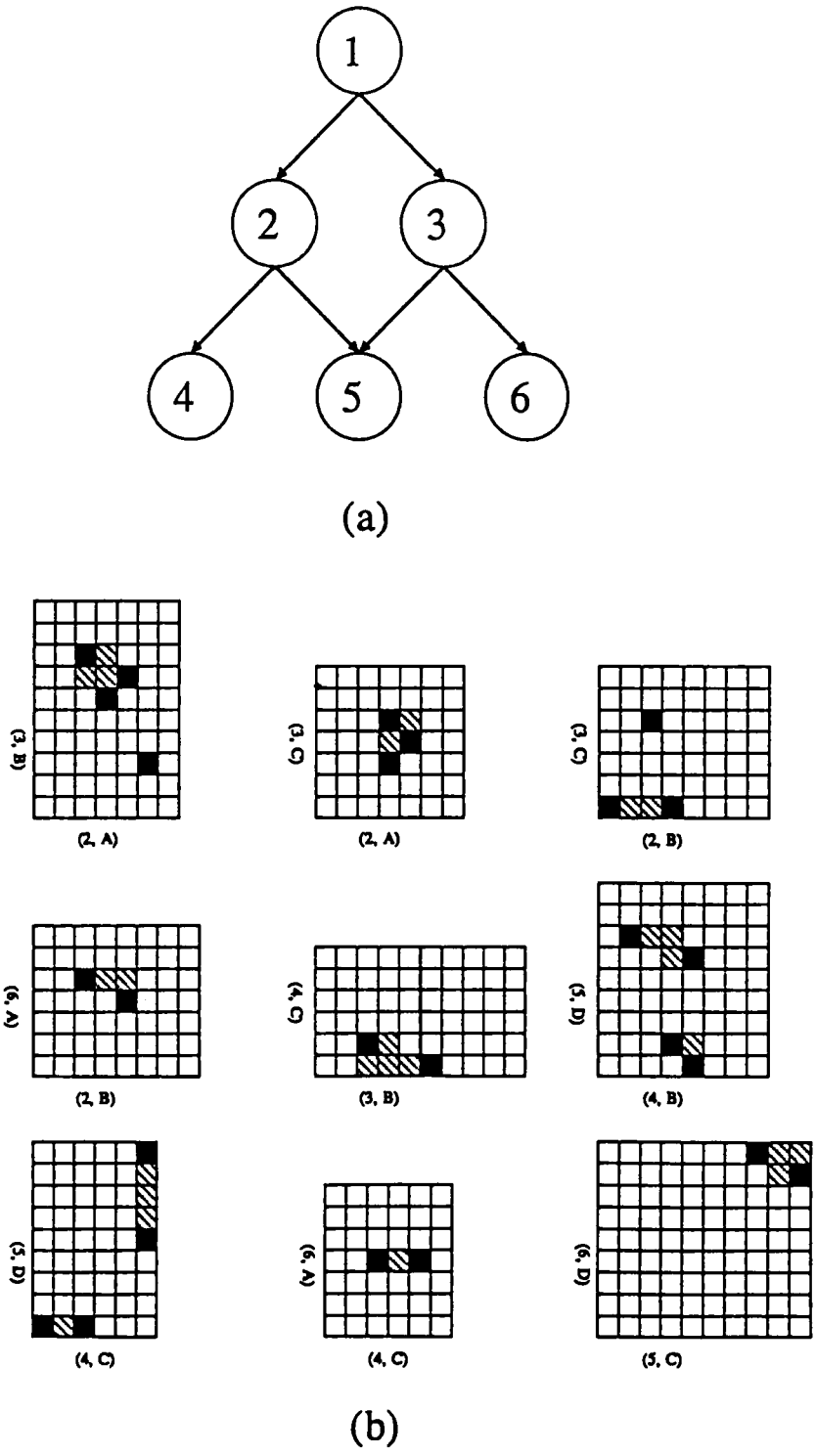
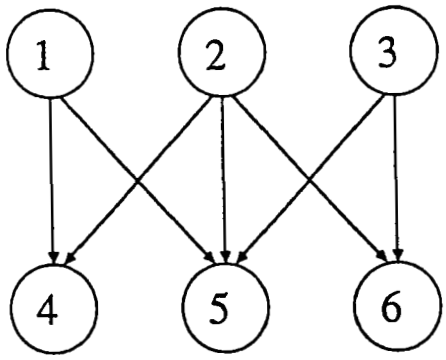


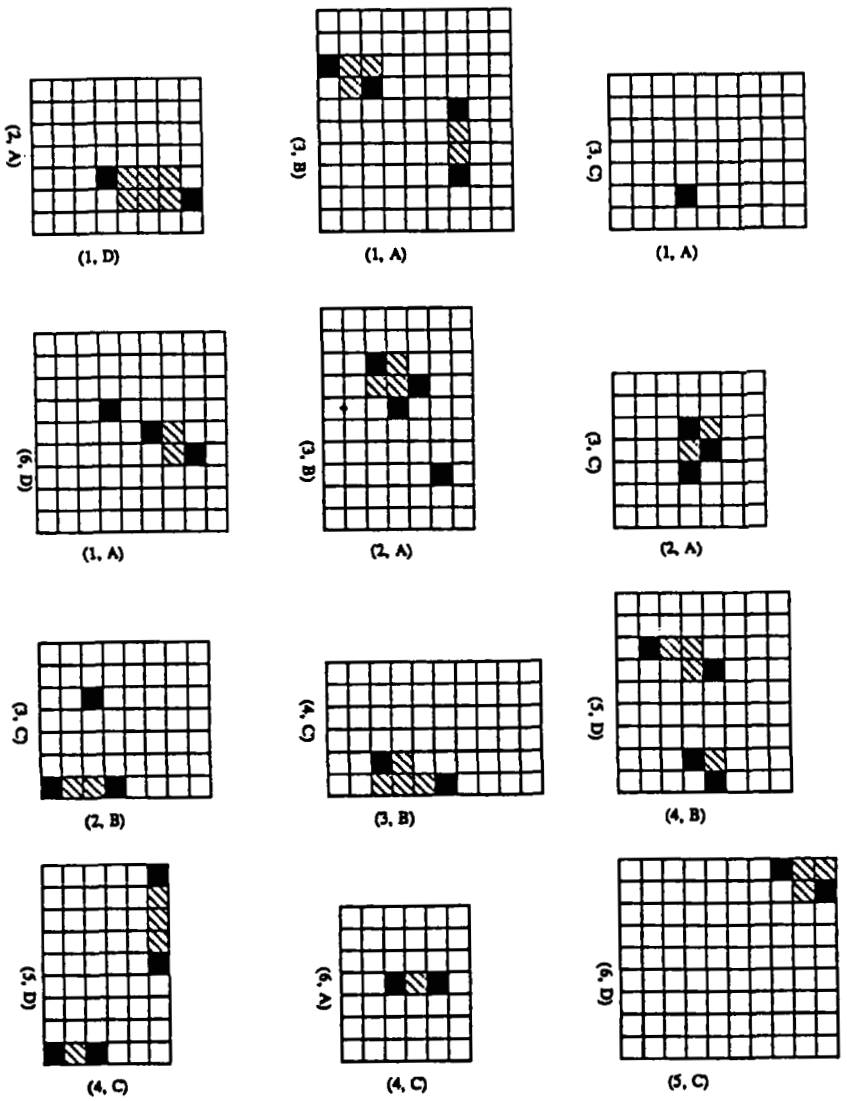
Figure 13. Example 2: (a) the task graph; (b) the schedule maps.

the tasks and the waiting time for avoiding robot collisions; and (3) an A^* search method of the optimal assignment with the minimum cost in the state-space tree.

Also proposed are a method to reduce the size of the state-space tree by eliminating the redundant tree paths from the tree, and some effective heuristic rules for quick search of the best goal node with the



(a)



(b)

Figure 14. Example 3: (a) the task graph; (b) the schedule maps.

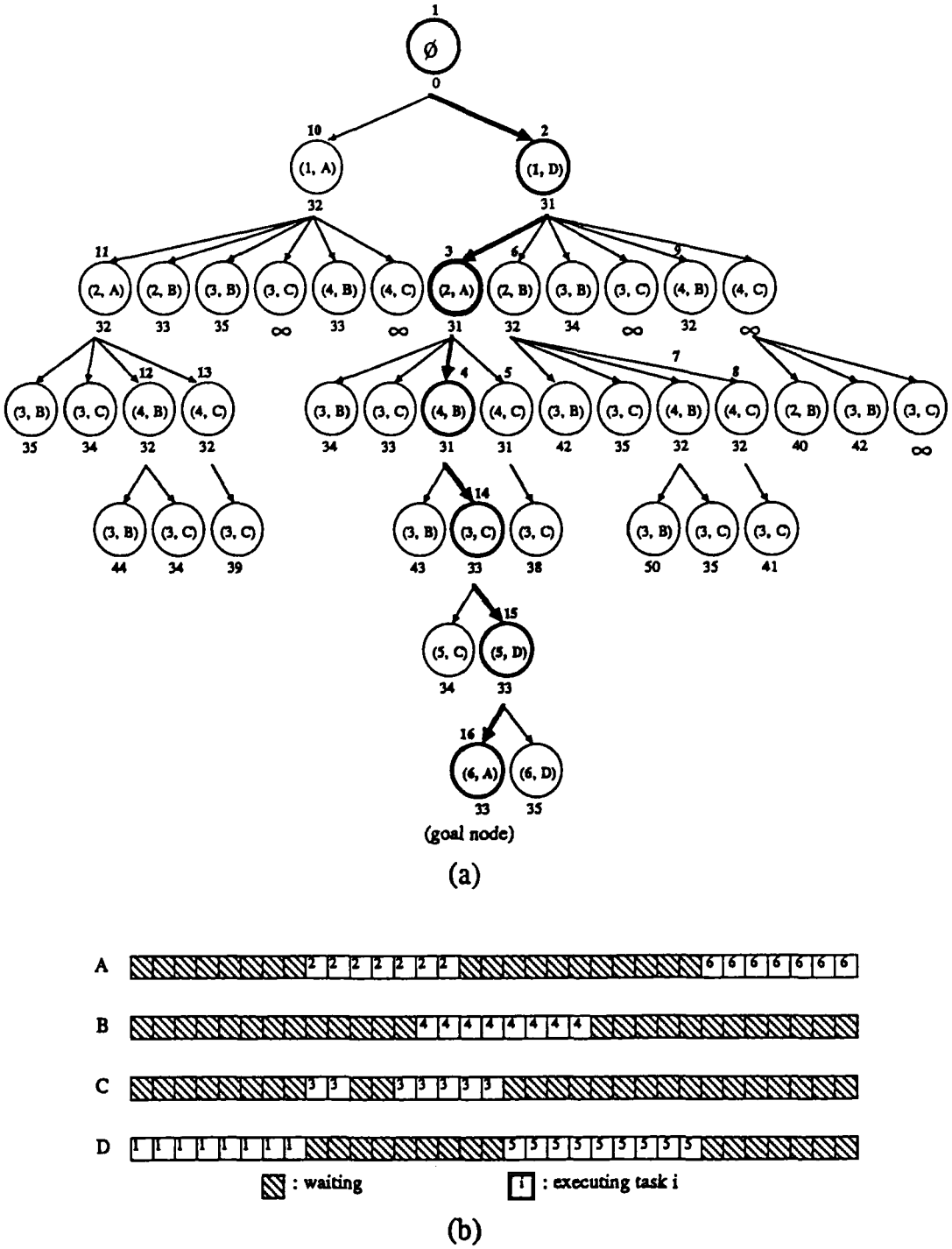


Figure 15. The state-space search tree and the scheduling results of Example 1 (the optimal solution path is shown dark): (a) the state-space search tree with a number above a node indicating node expansion order and a number below indicating $f(n)$ value; (b) the collision-free scheduling results represented by Gantt charts.

minimum cost. They improve the performance of finding the optimal assignment.

The precedence relationships of the tasks in a process plan are represented by a directed acyclic graph. However, not all process plans can be described by such graphs. In some cases the tasks may be described by AND/OR graphs.[15] Future research may be directed to handling such graphs in the pro-

posed approach. Conducting a practical implementation of the proposed approach on a real multi-robot system to prove the feasibility of the approach is also under consideration.

REFERENCES

1. O. Z. Maimon, "A generic multirobot control experimental system," *J. Rob. Syst.* **3**(4), 451–466, 1986.
2. O. Maimon, "The robot task-sequencing planning problem," *IEEE Trans. Rob. Autom.* **6**(6), 760–765, 1990.
3. T. Nagata, K. Honda, and Y. Teramoto, "Multirobot plan generation in a continuous domain: Planning by use of plan graph and avoiding collisions among robots," *IEEE J. Rob. Autom.* **4**(1) 2–13, 1988.
4. C. L. Chen, C. S. G. Lee, and C. D. McGillem, "Task assignment and load balancing of autonomous vehicles in a flexible manufacturing system," *IEEE J. Rob. Autom.* **RA-3**(6), 659–671, 1987.
5. L. L. Wang and W. H. Tsai, "Optimal assignment of task modules with precedence for distributed processing by graph matching and state-space search," *BIT*, **28**, 54–68, 1988.
6. C. F. Lin and W. H. Tsai, "Motion planning of multiple robots with multi-mode operations via disjunctive graphs," *Robotica*, **9**, 393–408, 1991.
7. C. C. Shen and W. H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion," *IEEE Trans. Comput.*, **C-34**(3), 197–203, 1985.
8. E. Horowitz and S.Sahni, *Fundamentals of Data Structures*. Woodland and Hills, CA, 1982.
9. E. Balas, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm," *Oper. Res.*, **17**(6), 941–957, 1969.
10. M. Florian, P. Trepant, and G. McMahon, "An implicit enumeration algorithm for the machine sequencing problem," *Manage. Sci.*, **17**, B782–B792, 1971.
11. S. Ashour, T. E. Moore, and K. Y. Chiu, "An implicit enumeration algorithm for the nonpreemptive shop scheduling problem," *AIIE Trans.*, **6**(1), 62–72, 1974.
12. C. F. Lin and W. H. Tsai, "Trajectory modeling, collision detection, and motion planning for two robot manipulators," *Int. J. Rob. Autom.*, **6**(4), 193–209, 1991.
13. R. J. Wilson and L. W. Beineke, *Applications of Graph Theory*. Academic Press, New York, 1979.
14. N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.
15. L. S. H. D. Mello and A. C. Sanderson, "And/or graph representation of assembly plans," *IEEE Trans. Rob. Autom.*, **6**(2), 188–199, 1990.