Proceedings of the 2000 IEEE
International Conference on Robotics & Automation
San Francisco, CA • April 2000

On-board Task Scheduling Algorithm for Spacecraft

Il-Jun Jeong[1], George Papavassilopoulos[1], and David S. Bayard[2]

[1] Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089

[2] Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract

An on-board scheduling algorithm is developed for scheduling tasks on a spacecraft. For offline scheduling, a heuristic routine generates a feasible schedule and order-based genetic algorithms improve the schedule. For online scheduling, two heuristic routines generate a good schedule in near real-time. The on-board scheduling algorithm is applied to a large problem of 1000 tasks, 100 resources and 365 time units. The results were successful: The on-board scheduling algorithm could generate various schedules according to time requirements.

1 Introduction

Developing on-board scheduling algorithms is essential for autonomous spacecraft, which can take over many mission operations that traditionally have been performed on the earth. Traditional spacecraft have used older proven technologies than newer unsafe ones, because spacecraft mission is too valuable to take risk. The objective of NASA's New Millenium Program (NMP) is to test and validate the new technologies for future space mission [5].

On-board scheduling enables spacecraft to be more responsive to unexpected opportunities and more robust to sudden failures. When spacecraft flies by a target to take a sequence of images, an unexpected important object may appear. On-board scheduling can utilize the opportunities, and produce a better outcome of space mission. Traditional ground scheduling approach could not react on component failures promptly, due to the communication delay between the spacecraft and the earth. The new approach is also less costly. The usage of ground station's resources to resolve the component failures, and network of antennas for the communication can be minimized. For example, the Deep Space One (DS1), which is the first mission of New Millenium Program (NMP), requires approximately one pass of Deep Space Network (DSN) coverage per week [5].

Task scheduling on a spacecraft is selecting and arranging tasks to maximize the successful outcome of a space mission while satisfying resource and temporal constraints. Tasks have different duration and resource demands. The resource demands can be constant or time varying. Payment values are assigned to tasks, and used to measure the quality of the outcome. Various resource types are used to represent resource constraints. Laser instruments can be represented by renewable resource constraint. Amount of fuel can be represented by a non-renewable resource constraint. A feasible schedule also must satisfy many temporal constraints, including precedence constraints (e.g. the engine should be warmed up before firing), non-overlapping constraints (e.g. a laser instrument cannot be used for two separate objects), and specific range constraints (taking images of an object should be done within specific time range).

The Hubble Space Telescope (HST) scheduling is another application of on-board scheduling algorithms. The HST scheduling need to process some 10,000 to 30,000 observation per year, and each observation is restricted by available observing time, computer storage, tape recorder storage, and amount of communication data [4]. The HSP scheduling is mostly offline scheduling, however online scheduling is also necessary. For example, a new important observing target, component failures, and changes in observing programs require online scheduling.

Some scheduling tools have been developed for spacecraft scheduling. For example, GERRY has been applied to the Space Shuttle Ground Operations Problem [8]. GERRY uses a constrained-based iterative repair method, which iteratively modifies a given complete but possibly infeasible schedule to improve the quality of the schedule. To avoid local minima and cycles, the system used simulated annealing algorithms while keeping the best schedule separately. Another example of the scheduling tools is SPIKE, which has been developed for the HST scheduling problem [4]. SPIKE uses a heuristic repair-based scheduling method, which selects the best of many trials of different initial guess and repair heuristics.

Our on-board scheduling algorithm consists of three routines: routine 1, routine 2, and the order-based genetic algorithm. Routine 1 removes infeasible tasks to generate a feasible schedule. Routine 2 adds feasible tasks to refine the schedule. The order-based genetic algorithm improves the schedule iteratively. To test on-board scheduling algorithm, we generate a problem consisted of 1000 tasks, 100 resources, and 365 time units. We tested the on-board scheduling algorithm under dynamic situation.

The remaining part of this paper is organized as follows: Section 2 defines problems to solve. We examine task, resource, and temporal constraints. We also discuss scheduling objective. Section 3 describes our on-board scheduling algorithm. We explain each routine of scheduling algorithm and each stage of scheduling. Section 4 shows experimental results. The experiments are intended to show how the scheduling algorithm performs on actual spacecraft scheduling situation. Finally, Section 5 presents conclusions.

2 Problem Description

A task i is characterized by duration d_i and resource demand r_{ik} on resource k. If the task i is scheduled to start at t_i, then the task i will be completed at $t = t_i + d_i$. We assume that the time t is an integer and $0 \le t \le T$, where T is an scheduling period. Sometimes, the task i can be performed in more than one mode. To include such case, we generalize the above symbols: a task i in mode m has duration d_{im} and resource demand r_{imk} on resource k.

Resources are categorized into three groups [6, 7]: renewable resources, non-renewable resources, and doubly constrained resources. Renewable resources are restorative, and have the maximum allowable limit on a period-by-period basis. Thus, a previously consumed amount does not need to be considered. Non-renewable resources are consumable. The maximum allowable limit is on a total period basis. The available amount of non-renewable resources depends on the amounts previously consumed. Doubly constrained resources have both characteristics of renewable and nonrenewable resources.

We consider three types of temporal constraints: precedence constraints, non-overlapping constraints, and specific range constraints. Precedence constraints confirm that some tasks are performed before others. If a task i should be finished before a task j, then:

$$t_j \ge t_i + d_{im} + \tau_{ij}$$

where τ_{ij} is a minimum time interval between the task i and the task j. Non-overlapping constraints restrict overlapping of two tasks. They can be expressed as:

$$t_i \ge t_j + d_{jm} + \tau_{ji} \quad \text{or} \quad t_j \ge t_i + d_{im} + \tau_{ij}$$

and, in this case, the task i and the task j cannot be executed at the same time. The $\tau_{ij} = \tau_{ji}$ is an additional time interval between the two tasks. Specific period constraints restrict the start time of a task i within a given period.

$$0 \le a \le t_i \le b \le T$$

For example, some observations need to be performed at a certain time due to the location of a spacecraft.

Because the time interval [0, T] of a space mission is predefined and limited, it may be impossible to select and schedule all intended tasks: We may need to reject some tasks. To distinguish rejected tasks and selected tasks, we use a parameter $e_i \in \{0, 1\}$: $e_i = 0$ indicates that a task i is rejected, and $e_i = 1$ indicates that the task i is selected. We assume that $\gamma_i > 0$ is the reward when the task i is selected and $\rho_i < 0$ is penalty when the task i is rejected. Then:

$$\gamma_i e_i + (1 - e_i)\rho_i = (\gamma_i + \rho_i)e_i - \rho_i = w_i e_i - \rho_i$$

where w_i is a payment of the task i.

We want to find a schedule $S(t_1,\ldots,t_n,e_1,\ldots,e_n)$ that maximizes the payment of all selected tasks while satisfying all constraints. In other words:

Problem: Find $e_i \in \{0, 1\}$ and $t_i \in [0, T\text{-}d_i]$, $i = 1, \ldots, n$, so that:

$$\max_{e_i} \sum_{i=1}^{n} w_i e_i \tag{1}$$

subject to: $S(t_1,\ldots,t_n,e_1,\ldots,e_n)$ satisfies all constraints.

When we can easily select all tasks or when we want to find the minimum makespan, we need to replace (1) with:

$$\max_{e_i} \sum_{i=1}^{n} w_i e_i + T - \max_i (t_i + d_i) \tag{2}$$

The T ensures that the result of (2) is positive. For multi-mode problems, (2) should be:

$$\max_{e_i} \sum_{i=1}^{n} w_{im} e_i + T - \max_i (t_i + d_{im}) \tag{3}$$

3. On-board Scheduling Algorithm

Figure 1 shows the block diagram of the on-board scheduling algorithm. Section 3.1 explains each routine of the scheduling algorithm, and Section 3.2 shows how the routines work for offline and online scheduling.

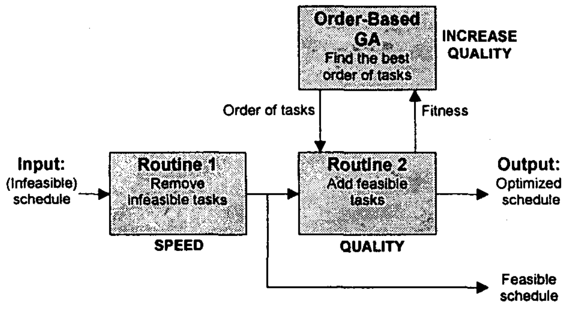
Order-Based GA Find the best order of tasks
INCREASE QUALITY
Order of tasks
Fitness
Input: (Infeasible) schedule
Routine 1 Remove infeasible tasks
SPEED
Routine 2 Add feasible tasks
QUALITY
Output: Optimized schedule
Feasible schedule

Figure 1. Block diagram of on-board schedulin algorithm

3.1 Routines of On-board Scheduling Algorithm

- *Routine 1*

The input of routine 1 is a schedule: The schedule could be a randomly generated one or previously optimized one. If the schedule is not feasible, routine 1 removes infeasible tasks from the schedule to generate a feasible schedule. The major role of routine 1 is generating a feasible schedule as soon as possible when a sudden change occurs. For example, if a very important observing opportunity appears during a space mission, the on-board scheduling algorithm has to rearrange tasks to accommodate the opportunity. Also, if some resources become unavailable because of some components' failure, on-board scheduling algorithm has to remove all tasks that require the resources. The algorithm of routine 1 is as follows:

```
for t = 1 : T
    for task = first_task : last_task
        if task is scheduled at t
            if task does not satisfy constraints
                remove task;
            end
        end
    end
end
```

- *Routine 2*

Routine 2 [3] takes an order of tasks as input and generates a feasible schedule. From the first time unit t = 1, the routine 2 checks whether the task, which is selected according to the given order, satisfies all given constraints. If the task satisfies the constraints, the routine places the task at the time unit. The above procedures continue until no more tasks are available (i.e. all tasks are selected) or the time reaches the end of an intended period. The algorithm of routine 2 is as follows:

```
for t = 1: T
    for task = first_task : last_task
        if task is not scheduled
            if task satisfies constraints
                add task at t;
            end
        end
    end
end
```

- *Order-based Genetic Algorithms*

The order of tasks determines the quality of a schedule. Thus, to increase the quality, we need to find better order of tasks. Order-based genetic algorithms are selected for the purpose. We selected PMX (Partially Mapped Crossover) [1] as crossover and SIM (Simple-Inversion Mutation) [2] as mutation.

The PMX tries to preserve the orders and positions of parents as many as possible. For example, consider the following two parents:

$$P_1 = [2\ 6\ 5\ 3\ 8\ 7\ 9\ 1\ 4],$$
$$P_2 = [9\ 5\ 3\ 4\ 8\ 2\ 1\ 6\ 7].$$

To generate two offsprings O_1 and O_2, the PMX randomly selects two cut points. Suppose they are the third and seventh intervals, which are indicated by '|':

$$P_1 = [2\ 6\ 5\ |\ 3\ 8\ 7\ 9\ |\ 1\ 4],$$
$$P_2 = [9\ 5\ 3\ |\ 4\ 8\ 2\ 1\ |\ 6\ 7].$$

The substrings between the two cut points are called the mapping segments. They defines the following mappings: $3 \leftrightarrow 4$, $8 \leftrightarrow 8$, $7 \leftrightarrow 2$, and $9 \leftrightarrow 1$. First, swap the mapping segments (the symbol 'x' represents 'currently unknown'):

$$O_1 = [x\ x\ x\ |\ 4\ 8\ 2\ 1\ |\ x\ x],$$
$$O_2 = [x\ x\ x\ |\ 3\ 8\ 7\ 9\ |\ x\ x].$$

Then, replace the 'x' with the elements of parents, if it is not conflicted with the elements of the mapping segment. The result will be:

$$O_1 = [x\ 6\ 5\ |\ 4\ 8\ 2\ 1\ |\ x\ x],$$
$$O_2 = [x\ 5\ x\ |\ 3\ 8\ 7\ 9\ |\ 6\ x].$$

Finally, replace the 'x' with the mappings that are defined previously. For example, the first element of O_1 should be

7, because the mapping 7 $\leftrightarrow$ 2. The final offsprings will be:

$$O_1 = [7\ 6\ 5\ |\ 4\ 8\ 2\ 1\ |\ 9\ 3],$$
$$O_2 = [1\ 5\ 4\ |\ 3\ 8\ 7\ 9\ |\ 6\ 2].$$

The SIM chooses a substring at random and reverses the elements of it. For example, if substring [5 3 8 7] is reversed from parent P = [2 6 5 3 8 7 9 1 4], offspring will be O = [2 6 7 8 3 5 9 1 4].

3.4 Stages of On-board Scheduling Algorithm

On-board scheduling algorithm has four stages. Stage 1 is for offline scheduling. Stage 2, 3 and 4 are for online scheduling.

- *Stage 1: offline scheduling (Figure 2)*

We emphasize the quality of a solution. Routine 2 generates a feasible schedule from the order of tasks, and the order-based genetic algorithm finds better order of tasks based on the quality of the schedule. The quality of the schedule serves as a fitness value for the genetic algorithm. This stage generally requires a long execution time, but we can use a powerful computer or several computers.

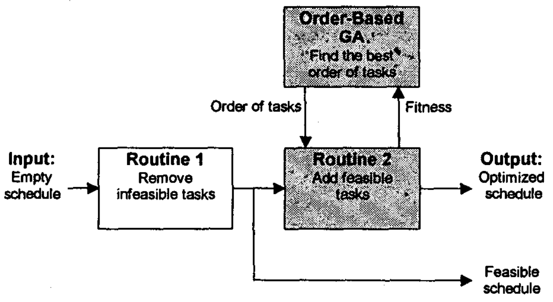
Order-Based GA Find the best order of tasks
Order of tasks
Fitness
Input: Empty schedule
Routine 1 Remove infeasible tasks
Routine 2 Add feasible tasks
Output: Optimized schedule
Feasible schedule

Figure 2. Stage 1 of on-board scheduling algorithm

- *Stage 2: online scheduling (Figure 3)*

We emphasize the speed of a solution. We use the previously optimized schedule and the order of tasks. If we need to execute a new important task at a time, we place the task at the time. Routine 1 removes any infeasible tasks from the placement and generates a feasible schedule. However, it is still possible that the task cannot be scheduled because of temporal constraints.

If some instruments are inoperative, the resource capacities of the instruments will suddenly change. In such case, the scheduling algorithm automatically removes infeasible tasks and generates a feasible schedule.

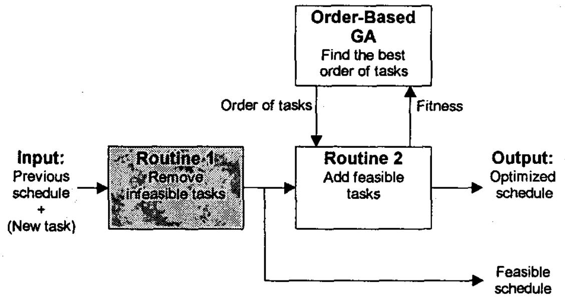
Order-Based GA Find the best order of tasks
Order of tasks
Fitness
Input: Previous schedule + (New task)
Routine 1 Remove infeasible tasks
Routine 2 Add feasible tasks
Output: Optimized schedule
Feasible schedule

Figure 3. Stage 2 of on-board scheduling algorithm

- *Stage 3: online scheduling (Figure 4)*

Routine 2 improves the schedule of Stage 2. We expect the improvement within a reasonable time. Two methods can be used: First, we simply add remaining tasks on the schedule of Stage 2. It is faster than the next method but the quality will be worse. Second, we totally reschedule except completed tasks and currently running tasks. We take the second method in this paper.

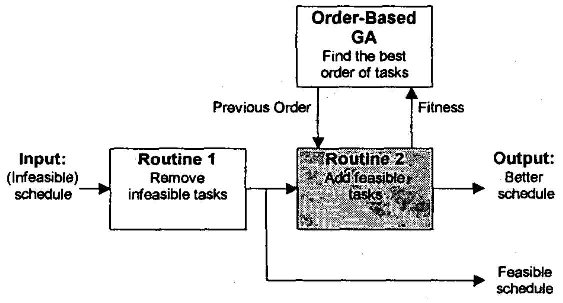
Order-Based GA Find the best order of tasks
Previous Order
Fitness
Input: (Infeasible) schedule
Routine 1 Remove infeasible tasks
Routine 2 Add feasible tasks
Output: Better schedule
Feasible schedule

Figure 4. Stage 3 of on-board scheduling algorithm

- *Stage 4: optional online scheduling (Figure 5)*

This stage improves the previous solution further by the order-based genetic algorithm. This stage is optional but recommended if major changes occur.

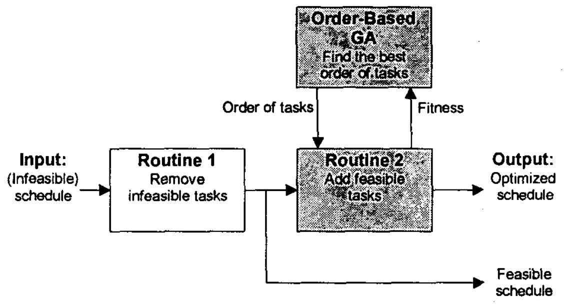
Order-Based GA Find the best order of tasks
Order of tasks
Fitness
Input: (Infeasible) schedule
Routine 1 Remove infeasible tasks
Routine 2 Add feasible tasks
Output: Optimized schedule
Feasible schedule

Figure 5. Stage 4 of on-board scheduling algorithm

4 Experimental Studies

The performance of the on-board scheduling algorithm is examined in this section. The performance of execution time is main interest. The initial test data is as follows: The number of tasks is 1000, and each task requires 50 simple renewable resources and 50 simple non-renewable resources. Each resource demand is a randomly generated integer between 0 and 10. The task duration is a randomly generated integer between 1 and 10. The payment of task *i* is calculated by:

$$w_i = d_i \cdot \sum_{k=1}^{100} r_{ik}$$

Scheduling period is 365 time units. Tasks has following precedence constraints:

$$t_1 + d_1 \leq t_2, t_2 + d_2 \leq t_3, \ldots, t_9 + d_9 \leq t_{10}$$
$$t_{11} + d_{11} \leq t_{12}, t_{12} + d_{12} \leq t_{13}, \ldots, t_{19} + d_{19} \leq t_{20}$$
$$\ldots\ldots$$
$$t_{91} + d_{91} \leq t_{92}, t_{92} + d_{92} \leq t_{93}, \ldots, t_{99} + d_{99} \leq t_{100}.$$

For the order-based genetic algorithms, the number of population is 10 and the number of generation is 50. The PMX is used for crossover and the SIM is selected for mutation. The crossover rate is set to 0.9 and mutation rate is set to 0.001.

The on-board scheduling algorithm is implemented using MATLAB. An IBM compatible PC (Intel Pentium 133Mhz processor and 32 Mb of main memory) is used for experiments.

4.1 Test 1: Sudden New Task

- *Stage 1*

The on-board scheduling algorithm searches the best order of tasks that generates an optimum schedule by an order-based genetic algorithm. After 50 generations, the scheduling algorithm found the order of tasks X_1 and a schedule S_1. The number of selected tasks of the schedule S_1 is 775 and the total payment is 2151672. The total search time was about 38 hours, which is quite long but this stage is usually performed before the launch of a spacecraft. To reduce the scheduling time, a simple parallel algorithm can be used. For example, each individual can be independently evaluated; if we have 10 computers, we can find the X_1 about 3.8 hours.

- *Stage 2*

At time = 100, one new task was added for immediate execution: This was done by setting start time = 100 and level = 0 (the highest level) for the task. First, our scheduling algorithm removed all uncompleted job, whose finish time is greater than 100, and placed the new task. Then, the on-board scheduling algorithm placed all feasible tasks according to the schedule S_1. The new schedule S_2 included total 757 tasks and had total payment 2094941. The execution time was about 15 seconds.

- *Stage 3*

On-board scheduling algorithm added remaining unscheduled tasks onto the schedule S_2 according to the order of tasks X_1. The new schedule S_3 included total 775 tasks and had total payment 2134544. The execution time was about 230 seconds.

- *Stage 4*

Order-based genetic algorithm improves the schedule S_3. Order-based genetic algorithm uses the same number of population and generation. Because the range of scheduling period is between 100 and 365, the execution time was shorter about 24 hours 30 minutes. The total payment of stage 4 is lower than the payment of stage 1. It is reasonable because there will be some losses due to removal of tasks that were not completed before 100. The results of above stages are summarized in Table 1.

	Execution Time	Number of Selected Tasks	Total Payment
Stage 1	38 hours	775	2151672
Stage 2	15 seconds	757	2094941
Stage 3	230 seconds	775	2134544
Stage 4	24.5 hours	775	2140243

Table 1. Result of test 1

4.2 Test 2: Unexpected Resource Failure

- *Stage 1*

This is same as above case: The number of selected tasks of the schedule S_1 was 775 and the total payment was 2151672. The total search time was about 38 hours.

- *Stage 2*

At time = 200, the capacity of one renewable resource was reduced to half. The on-board scheduling algorithm placed all feasible tasks according to the schedule S_1. The new schedule S_2 included total 618 tasks and had total payment 1721898. The execution time was about 9 seconds.

- *Stage 3*

The on-board scheduling algorithm added remaining unscheduled tasks onto the schedule S_2 according to the order of tasks X_1. The new schedule S_3 included total 676 tasks and had total payment 1886221. The execution time was about 120 seconds.

- *Stage 4*

The order-based genetic algorithm improved the schedule S_3. It took about 12 hours 30 minutes for 10 population and 50 generations. The improvement from the results of Stage 3 was noticeable, because there were significant changes on data set. The results of above stages are summarized in Table 2.

	Execution Time	Number of Selected Tasks	Total Payment
Stage 1	38 hours	775	2151672
Stage 2	9 seconds	618	1721898
Stage 3	120 seconds	676	1886221
Stage 4	12.5 hours	705	1942539

Table 2. Result of test 2

5 Conclusions

The performance of the on-board scheduling algorithm is demonstrated on a problem having 1000 tasks, 100 resources, and 365 scheduling period. Stage 1 of on-board scheduling algorithm took 38 hours for 50 generations of 10 individuals. The scheduling time can be greatly reduced if each individual is evaluated on separate computers. Stage 2 generated a feasible schedule about 15 seconds for Test 1 and about 9 seconds for Test 2. Stage 3 improved the feasible schedule within a reasonable time: 230 seconds for Test 1 and 120 seconds for Test 2. Stage 4 searched an optimum solution iteratively. The results of each stage can be used according to situation: If a feasible schedule is required urgently, the result of stage 2 or stage 3 has to be used; if there is enough time, the result of stage 4 will be desired.

Acknowledgments

We are grateful to David H. Collins and William "Curt" Eggemeyer of the Jet Propulsion Laboratory for helpful discussions, and to Hamid Kohen of JPL for additional software support on this project. The research described in this document was carried out for the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] D. E. Goldberg and J. R. Lingle, "Alleles, loci, and the traveling salesman problem," *Proceedings of the First International Conference on Genetic Algorithms and Their Applications,* Pittsburgh, PA, pp. 154-159, 1985.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems,* Ann Arbor, MI, The University of Michigan Press, 1975.

[3] I. Jeong, G. Papavassilopoulos, and D. S. Bayard, "Task Scheduling on Spacecraft by Hybrid Genetic algorithms," *Proceedings of the IEEE International Conference on Robotics & Automation,* Detroit, MI, pp. 441-446, 1999.

[4] M. D. Johnston and G. E. Miller, "SPIKE: Intelligent Scheduling of Hubble Space Telescope Observations," in *Intelligent Scheduling,* M. Zweben and M. S. Fox, Ed., Morgan Kaufmann, pp. 391-422, 1994.

[5] N. Muscettola, C. Fry, K. Rajan, B. Smith, S. Chien, G. Rabideau, and D. Yan, "On-Board Planning for New Millennium Deep Space One Autonomy," *Proceedings of IEEE Aerospace Conference,* Snowmass, CO, 1997.

[6] R. Slowinski, "Multiobjective network scheduling with efficient use of renewable and nonrenewable resources," *European Journal of Operational Research,* vol. 7, pp. 265-273, 1981.

[7] J. Weglarz, "Project scheduling with discrete and continuous resources." *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 9, pp. 644-650, 1979.

[8] M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and rescheduling with iterative repair," in *Intelligent Scheduling,* M. Zweben and M. S. Fox, Ed., Morgan Kaufmann, pp. 241-255, 1994.