A Planning/Scheduling Methodology for the Constrained Resource Problem[1]

Naiping Keng and David Y. Y. Yun
Department of Computer Science and Engineering
Southern Methodist University
300C Science Information Center
Dallas, Texas 75275

Abstract

This paper presents a new planning/scheduling methodology for the constrained resource problem *(CRP),* in which the amount of available resources is limited and usually monotonically diminishing as the planning process progresses. The tasks are tightly-coupled since they compete for the limited resources. Two domain independent policies — *most-constraint* and *least-tmpact* help to make this planning/scheduling approach sensitive to dynamic interactions among tasks. The most-constraint policy selects a task dynamically according to the criticality, which measures how a task is constrained by task interaction. The least-impact policy dynamically chooses a solution for the selected task according to the cruciality of each possible solution, which expresses the impact on the rest of the unachieved tasks. These policies have enhanced the operability and measurability of problem solving by planning/scheduling. Hence, this method can provide realistic and executable planning/scheduling guidelines for *CRP* solvers. This model has been successfully applied to several *CRPs* in which the amount of backtracking has been reduced dramatically.

1 Introduction

Many artificial intelligence tasks can be formulated as the constraint-satisfaction problem *(CSP)* which involves the assignment of values to variables subject to a set of constraints. The *CSP* consists of a finite set of n variables (or tasks) $V_1, \ldots, V_n$, a set of domains $D_1, \ldots, D_n$ and a set of constraint relations $C_1, \ldots, {}_1$. Each D_i defines a finite set of values (or labels or solutions) that variable V_j may be assigned. A constraint $C_{i;}$ specifies the consistent or inconsistent choices among variables and is defined as a subset of the Cartesian product:

$$\underline{C_j} \subset D_1 \times D_2 \times \cdots \times D_n$$

In this paper the goal of the *CSP* is to find one tuple from $D_1 \times \cdot \quad \times D_n$ such that n assignments of values to variables satisfy all constraints simultaneously.

In the process of solving a *CSP,* two decisions have to be made at each cycle, i.e. which variable to instantiate next and which value to assign to that variable. Since heuristics are used to guide these decisions in solving difficult *CSPs,* backtracking is usually unavoidable. If the cost of backtracking is high, the problem of how to reduce the amount of backtracking become crucial. Intelligent backtracking such as dependency-directed backtracking in truth maintenance systems [Doyle, 1979] is one way to deal with the problem. For problems in which the dependency relation is complicated, however, dependency-directed backtracking may be ineffective. An example that illustrates this behavior is N-queens problem in which every decision depends on all previous decisions. An alternative to dependency-directed backtracking is to give good advice for each decision so that the number of backtracks can be minimized. Various strategies for this have been developed. For variable selection, Bitner, 1975, Haralick *et* a/., 1980, Purdom, 1983] try to order uninstantiated variables dynamically in every branch of the search tree so that the next variable is always the one that will most constrain the rest of the search tree. Usually the variable that has the least number of values left is selected. In value assignment, Dechter [1988] attempts to assign a value that maximizes the number of options available for future assignment. Dechter identifies classes of constraint graphs lending themselves to backtrack-free solutions and devises efficient algorithms for solving them. By selectively deleting constraints from the original constraint graph, the original problem is then transformed into a simplified and backtrack-free problem. The number of consistent solutions in the simplified problem is therefore used to establish priorities of value assignments to variables in the original problem.

A subclass of *CSP* — the *constrained resource problem (CRP)* — supplies significant information that may be used to guide the search. This information has

[1] This research was sponsored in part by the Defense Advanced Research Project Agency under contract MDA903-85-0182.

not been used in previous approaches. In *CRP,* variables are tasks and values are resources. In this framework, two important characteristics of the problem are *limited* resources and *non-sharable* resources, i.e. a value can only be assigned to one variable and the total number of values is limited. The N-queens problem is an example of *CRP* if we assume that the rows are tasks and the columns are resources. In *CRP* the tasks are competing for the limited resources, i.e. the tasks interact. These tasks have to work cooperatively in order to find the solution for the problem. Both the task selection and the resource assignment should be guided in the way that:

(1) the possibility of success in the future is maximized;

(2) the amount of backtracking is minimized.

In order to achieve these goals we require some measures for how task competition affects individual tasks and we must use these measurements to guide the decisions.

In this paper we propose an approach for *CRP* that is different from Dechter's and others. We demonstrate the approach and show its effectiveness on the N-queens problem. We then generalize the approach to solve the general *CRP* in which the solution for a task may contain more than one resource.

2 A Planning/Scheduling Methodology for CRP

The *CRP* consists of a finite set *T* of n task $T_1, \ldots, T_n$, a set *D* of domains (or solution space) $D_1, \ldots, D_n$, a finite set *R* of resource units $r_1, \ldots, r_m$, and a set *C* of constraint relations $C_1, \ldots, C_l$. Each D, defines a finite set of solutions $P_{iq}, \ldots, P_{it}$ that task T_i may be assigned. Each P_{ik} can satisfy T, and consists of a resource unit r_k.

Due to the heavy interactions among tasks in *CRP,* the way in which resources are assigned to a task greatly impacts the possible resource assignment of other tasks. The problem solver searching for the local optimal solution without considering the global consequence of the solution suffers from:

(1) myopia with respect to only a specific task interaction;

(2) premature commitment of resources to the task, thereby constraining the future possible resource assignments of the remaining tasks and the final solution.

Myopia and premature commitment often arise from implicitly or unnecessarily predetermined orderings on search control, rule firing, resource priority or ignorance of global interactions in the problem. They increase the chance of making wrong decisions at the early stage of the planning/scheduling process, consequently increasing the amount of backtracking and the overall cost of problem solving.

The traditional least commitment strategy, as used in the current literature [Sacerdoti, 1975, Tate, 1977, Stefik, 1981], attempts to make a commitment only if there is a clear reason for doing so. The *CRP* is underconstrained and there is usually more than one alternative for the direction in which to proceed. The least commitment strategy is insensitive to these distinct needs, and is hence incapable of dynamically handling the interactions, in a situation-specific manner. Therefore, in order to provide the flexibility of dealing with task selection separately from solution selection, while continuing to monitor and handle the interactions between the two decision phases, we propose two domain independent policies - most-constraint and least-impact - to govern task selection and resource assignment respectively. The most-constraint policy tries to select the task which has least chance to survive under the resource competition. The least-impact policy attempts to select a resource which needs the least cost to use up that resource. These policies can avoid myopia and premature commitment by doing situation-specific reasoning that is sensitive to task interactions. The task interactions, in turn, are modeled by considering the criticality of the task, the cruciality of the solution, and constraint propagation, as described below. The planning/scheduling methodology can then be represented by a "four-corner" model shown in Figure 1. The planner/scheduler repeats the following four steps. Starting with a current agenda,

a Select the most critical task by the most-constraint policy.

b. Formulate the solution space (feasible solutions) for the selected task.

c. Select the least crucial solution from the current solution space by the least-impact policy.

d. Commit the resource to the selected task and propagate constraints (generating a new agenda and updating the status of the tasks and the resources).

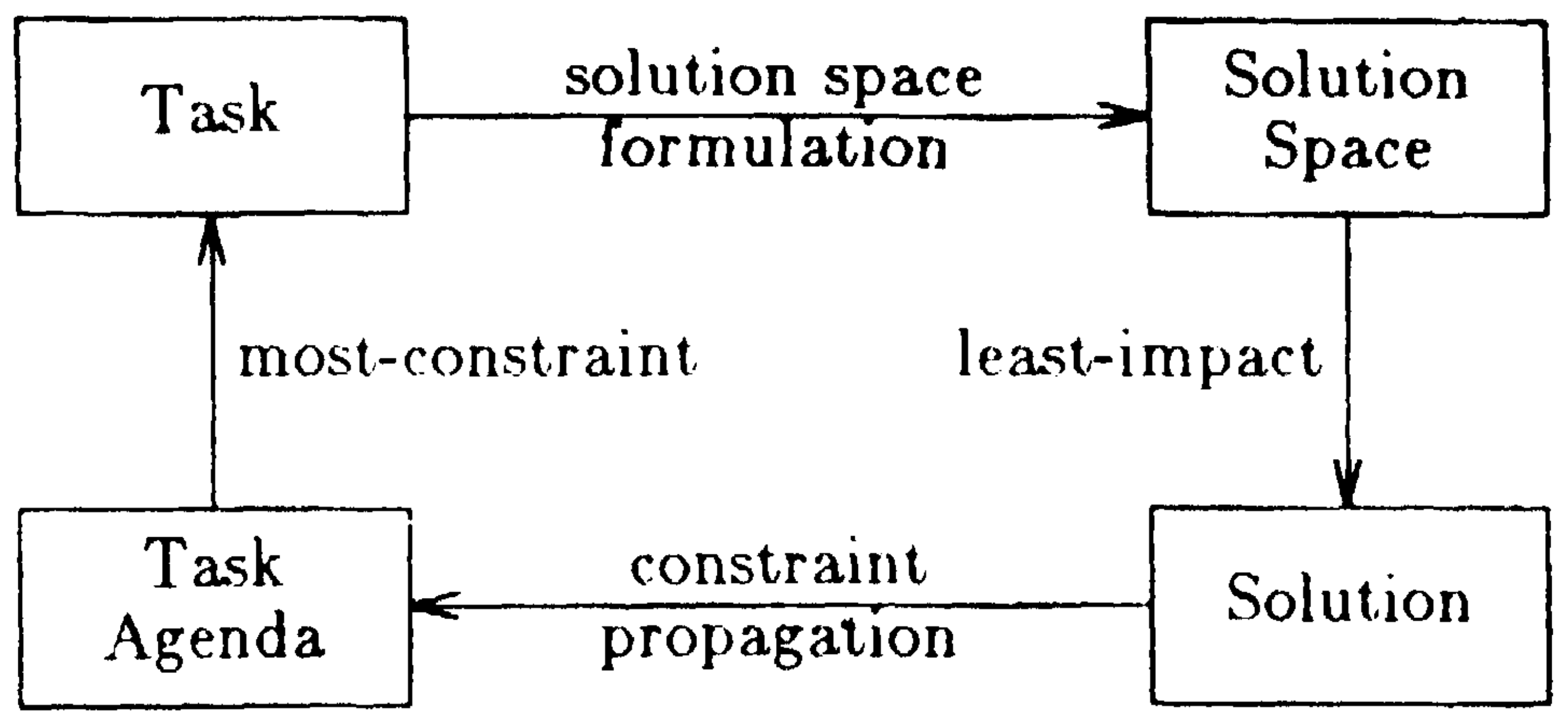


Figure 1 The Planning/Scheduling Model

2.1 Criticality of the Task

Each task is associated with a value called *criticality* which measures the impact of task interactions. The criticality of a task considers not only the number of possible resource assignments for the task, but also the impact from other tasks that are competing for the same resources. Task criticality measures the degree of flexibility for a task, i.e. the degree of survivability for a task that is constrained by resource competition and earlier commitments. The higher the criticality, the more flexible the task, and therefore the less critical the task. The most-constraint policy selects as the next task the most critical one. Task criticality is updated at each cycle since each commitment for a task reduces the number of possible resource assignments of the remaining tasks, due to the diminishing resources and the constraint relations. Hence, the criticality for each task is sensitive to any previous commitments.

Let the operation $ASSIGN(T_i, r_k)$ represent the attempt of assigning r_k to T_i. Let α_{ik} be the degree of need (or demand or competition) for r_k by T_i. Therefore, $\alpha_{ik} = 1 / |D_i|$ for all $r_k \in D_i$. Let γ_{ik} represent the strength of the impact/competition to $ASSIGN(T_i, r_k)$ from all unachieved tasks. Thus, $\gamma_{ik} = \alpha_{ik} + \sum \alpha_{jl}$, where T_j has possible resource assignments $ASSIGN(T_j, r_l)$ which is in conflict with $ASSIGN(T_i, r_k)$. That is, all unachieved tasks express their degree of dislike to the idea $ASSIGN(T_i, r_k)$ or their strength of the need/competition for r_k. Accordingly, $\alpha_{ik} / \gamma_{ik}$ represents the survivability for $ASSIGN(T_i, r_k)$ from the competition. Therefore, $\Omega_i = \sum_k (\alpha_{ik} / \gamma_{ik})$ where Ω_i represents the criticality of T_i, with respect to all possible assignments r_k to T_i, $r_k \in D_i$. The task having smallest criticality is the most constrained task and is selected as the next job. The solution space of the selected task T_i is the current corresponding domain D_i.

2.2 Cruciality of the Solution

Each individual solution is associated with a value called *cruciality* that measures the impact to the rest of the tasks if we commit a resource to a task. The cruciality of a solution is the cost of using up the resource in the solution. That is, in order to commit a resource to a task, we may have to give up several possible solutions for other tasks. The higher the cruciality of a solution, the more competitive the resource unit in the solution. The least-impact policy selects the solution having least cruciality for the chosen task, hence the commitment of that solution has a minimal impact on the remaining tasks. The cruciality of the solution is updated at each cycle because each commitment for a task may disable a resource unit, consequently increasing the criticalities of the affected tasks and changing the solution crucialities of unachieved tasks. The cruciality of the solution, therefore, is not only sensitive to the resource demands of unachieved tasks, but also sensitive to any commitment made earlier.

For the current task T_i, let Γ_{ik} represent the cruciality of the solution P_{ik} which contains r_k. $\Gamma_{ik} = \gamma_{ik} - \alpha_{ik}$, where γ_{ik} and α_{ik} has been defined above. That is, the commitment of $ASSIGN(T_i, r_k)$ is on the cost of reducing possible solutions of remaining tasks, i.e. reducing the flexibility of some remaining tasks. The solution having smallest cruciality has the least impact on the unachieved tasks and is hence committed to T_i.

2.3 Constraint Propagation

At each planning/scheduling cycle, the current environment should be consistent, i.e. no constraints should be violated and all unachieved subgoals should have at least one potential solution. Committing a solution, however, imposes new constraints (e.g. disables some resources) on the current environment. The current environment, therefore, has to be updated in order to reflect these changes. (One special case, called the *domino effect,* occurs when the impact of committing one task prunes another task's solution space to an unique solution.) Accordingly, a new consistent environment is generated or an inconsistency is found, which means a task is overconstrained. In that case, we retreat to the most recent choice point for which alternatives exist and make a new choice (i.e. retreat counterclockwise along the four-corner model).

3 An Example: the N-queens problem

We formulate the N-queens problem as a N x N board. Each position in the board is denoted by *ASSIGN(Ri,ck)* where R, and c_k represent the ith row and the kth column respectively. The tasks of the problem are rows and the resources are columns. The goal is to find a N tuple of consistent positions for N queens. Let D_t be the domain of row R_i where D_i, contains all possible column assignments to R_i. For all $c_k \in D_i$, $\alpha_{ik} = 1 / |D_i|$ and hence $\gamma_{ik} = \sum \alpha_{jl}$ where $l = k$ (same column) or $|i - j| = |k - l|$ (diagonal). Therefore, $\Omega_i = \sum_k (\alpha_{ik} / \gamma_{ik})$ where Ω_i is the criticality of R_i and $c_k \in D_i$. $\Gamma_{ik} = \gamma_{ik} - \alpha_{ik}$ where Γ_{ik} is the cruciality of $ASSIGN(R_i, c_k)$

We have implemented the proposed approach in C on a SUN 3/260 running 4.2 BSD Unix and tested it on board sizes ranging from N==4 to N==100. We compare the results with those generated by an approach which uses two simple heuristics to guide the row selection and column assignment. The heuristic for the row selection chooses the row having least number of possible column assignments. The heuristic for column assignment selects the column whose assignment maximizes the number of possible column assignments for remaining rows. For both approaches, ties are broken via the natural ordering of the indices. Figure 2(a) shows the number of backtracks as function of the problem size. The small o's and x's represent the numbers of backtracks generated by our approach and simple heuristic method, respectively. Figure 2(b) plots the execution time as functions of the problem size. The dashed line denotes the execution time of the program using the simple heuristic method, while the solid line indicates the execution time using our approach.

Using our approach, no backtracking was performed 76% of the time, the maximum number of backtracks in any single case was 13, and the average number of backtracks was less than 1. This indicates that our approach is usually successful in guiding the search in the right direction, especially at the early stages of the process. For the simple heuristic method, backtracking was performed 80% of the time, the maximum number of backtracks was more than 10^6, and there were 4 cases whose numbers of backtracks were more than 10^4. This indicates that the performance of the simple heuristic method is very unstable. In contrast, the performance of the proposed approach is consistent in the test cases. The main reason is that the number of possible column assignments for each row does not express the actual flexibility of the row. That is, the simple heuristic method makes decisions based on insufficient information that does not consider the impact of resource competition. Consequently, in several cases, it made wrong decisions at the very early stages of the process and thus, the simple method had long execution times due to the extremely large numbers of backtrack.

Our approach incurs considerable overhead in computing task criticality and solution cruciality. The number of backtracks is diminished to such an extent, however, that our approach is frequently competitive (and occasionally outperforms) the simple heuristic, even when backtracking is very cheap, as in the N-queens problem. For problems where backtracking is expensive, the strength of our approach is likely to be demonstrated more persuasively. There are, however, opportunities for reducing this overhead significantly by parallelizing these computations since all the computations for task criticalities and solution crucialities are independent.

4 Model Generalization

Our model can be generalized to more general class of *CRP* in which a task may demand more than one resource unit. That is, each D_i defines a finite set of subplans $P_{i1}, \ldots, P_{it}$ E very P_{ij} sists of a different set of resource units $\{r_k | r_k \in R\}$.

Let β_{ik} be the number of solutions including resource unit r_k in D_i, i.e. the importance o r_k t o T_i is increasing while the number of solutions that need r_k is increasing in D_i. T h $\alpha_{ik} = \beta_{ik} \quad |D_i|$ and $\gamma_{ik} = \alpha_{ik} + \sum \alpha_{jl}$ where *ASSIGN*(T_j, r_l) is in conflict with *ASSIGN*(T_i, r_k) due to resource competition or the constraint relation. Let ω_{ij} be the criticality of solution P_{ij}. $\omega_{ij} = \underset{k}{Min} \frac{\alpha_{ik}}{\gamma_{ik}}$ where $r_k \in P_{ij}$. Since a solution may contain more than one resource unit, we define the criticality of the solution to be the survivability of the most competitive resource unit. Therefore, the criticality of the task is the sum of the criticalities of their possible solutions, i.e. $\Omega_i = \sum \omega_{ij}$.

For selected T_i, solution cruciality is the impact of committing P_{ij} to other tasks. $\Gamma_{ij} = \sum_k (\gamma_{ik} - \alpha_{ik})$, where $r_k \in P_{ij}$. With these modified definitions of task criticality and solution cruciality, our planning/scheduling model can be applied to the general *CRP.*

5 Other Applications

We have attempted to apply our model to several *CRPs* such as VLSI switchbox routing and job-shop scheduling. In this paper we only formulate the *CRP* model for these two problems. Our executable model can then be applied to these problems according to the definitions described above.

5.1 Switchbox Routing Problem

Switchbox routing [Burstein *et* a/., 1983, Ho *et* a/., 1985] is an important task in the physical design of VLSI chips. The problem is formulated on a rectangular-grid with fixed-position net terminals on each of the four edges. Each terminal is identified by the name of the net to which it belongs and each net contains at least two terminals. A track is a line segment between two grid points and there are two routing planes. One plane contains horizontal tracks and the other one contains vertical tracks. Connection between a vertical track and a horizontal track is made through a small hole called via. Different nets cannot contact one another on the same layer. The task for the switchbox router is to connect all of the terminals with the same net name.

In switchbox routing, tracks and vias are resource units, while tasks correspond to the connecting of pairs of terminals. For each pair, the coordinates of the two terminals define four edges of a rectangle, i.e. *demand block (DB),* in which the two terminals are two opposite vertices. Each pair demands tracks and vias in the rectangle. The solution space of a pair contains all shortest paths connecting the pair in the rectangle. A path consists of at least one track. The application of our model, as well as the implementation and results have been presented in [Ho *et* a/., 1985, Keng *et* a/., 1987].

5.2 Job-Shop Scheduling Problem

The job-shop scheduling problem [Baker, 1974, Fox, 1983, Keng *tt* a/., 1988] consists of a set of machines and a set of orders of products. For each product, a partial ordering (process routing) of operations whose execution results in the completion of the product is given. For each order, the scheduler assigns a time interval (start and end times) and a machine to each operation in its corresponding process routing. The task for the job-shop scheduler is to construct a schedule in which all orders are finished.

The set of resource units contains (machine, time) pairs $\{(M_1, t_{11}), \ldots, (M_m, t_{mn})\}$, where t_{ij} denotes the time unit j on machine Mj-. Each machine is associated with a time horizon that is divided into consecutive time units The set of tasks are operations that are obtained through decomposition of the set of orders and their corresponding process routings. The precedence relations among operations are constraints. For each operation we can compute the resource demand, i.e. *demand window (DW),* that defines the earliest possible start-time and the latest possible end-time for the operation, using the ready time and the due date of the corresponding order. *DW* defines all possible time intervals that can be assigned to the operation and is the solution space of the operation. A time interval consists of at least one resource unit. A detailed discussion of the application of our model to the job-shop scheduling problem is found in (Keng *et ai,* 1988].

6 Conclusion

A dynamic planning/scheduling methodology for the *CRP* has been presented in this paper. Our model uses two domain independent policies — *most-constraint* and *least-impact* — to control the search for a solution. The most-constraint policy controls the selection of the next task according to the task *criticality* that measures how task competition for resources constrains a task. For a selected task, the least-impact policy guides the search for the solution according to the solution *cruciality,* that measures how the commitment of a solution affects the possible resource assignments for other tasks. The criticality and the cruciality are sensitive to the dynamically changing environment, because they are updated after each resource commitment and rearranged according to their new priorities. Consequently, our model acts opportunistically and in a constructive, situation-specific manner, without making premature commitment.

Experiments indicate that our approach is usually successful in guiding the search in the correct direction. In a set of 97 test cases using the above model, no backtracking was performed 76% of the time and the maximum number of backtracks in any single case was 13. Our model is likely to be highly appropriate for problems where the cost of backtracking is high. Our approach does incur considerable overhead in performing the analysis required for decision making, however there are opportunities for reducing this cost by parallelizing the computations of task criticality and solution cruciality.

The problems to which our model has been successfully applied includes the VLSI switchbox routing problem [Ho *ct ai,* 1985, Keng *ct ai,* 1987], the job-shop scheduling problem [Keng *ct ai,* 1988], the task assignment problem in parallel processing [Tsai, 1988]. By using two control policies and uniform representation to measure the resource competition among tasks, our methodology not only has provided realistic and executable planning/scheduling guidelines for *CRP* problem solvers, but also has enhanced the operability and measurability of problem solving by planning/scheduling. Hence, we believe our model presents a general methodology for problem solving by planning/scheduling.

References

Baker, 1974] K. R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons., New York, 1974.

[Burstein *ct ai,* 1983] M. Burstein and R. Pelavin. Hierarchical Wire Routing. *IEEE Trans. on Computer-Aided Design,* CAD-2(4):223-234, October 1983.

[Bitner, 1975] J. R. Bitner and E. M. Reingold. Backtrack Programming Techniques. *Communications of the ACM,* 18(II):651-656, 1975.

Dechter, 1988] R. Dechter and J. Pearl. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence,* 34:1-38, 1988.

[Doyle, 1979] J. Doyle. A Truth Maintenance System. *Artificial Intelligence,* 12(3):231-272, 1979.

[Fox, 1983] M. S. Fox. Constraint-Directed Search: A Case Study of Job-Shop Scheduling. Technical Report CMU-RI-TR-83-22, Carnegie-Mellon University, 1983.

[Haralick *ct ai,* 1980] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence,* 14(3):263-313, 1980.

Ho *ct ai,* 1985] W. Ho, D. Y. Y. Yun, and Y. H. Hu. Planning Strategies for Switchbox Routing. In *Proceedings of the International Conference on Computer Design,* pages 463-467, 1985.

Keng *et ai,* 1987] N. P. Keng, W. P.-C. Ho, and D. Y. Y. Yun. The Manual of the Switchbox Router. Technical Report 87-CSE-2, Southern Methodist-University, January 1987.

[Keng *et ai,* 1988] N. P. Keng, D. Y. Y. Yun, and M. Rossi. Interaction-Sensitive Planning System for Job-Shop Scheduling. *Expert Systems and Intelligent Manufacturing,* M. OlifT, editor, pages 57-69, 1988.

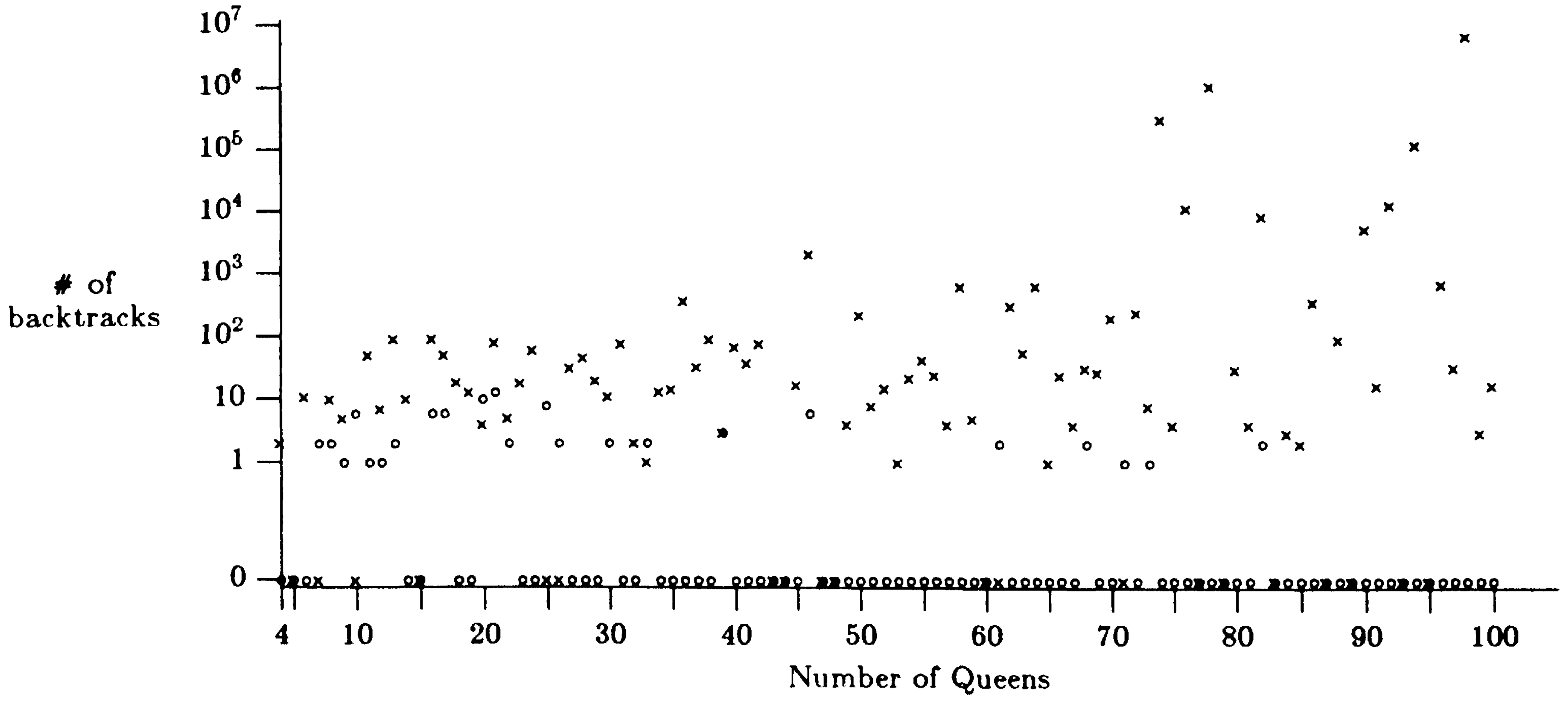
[Purdom, 1983] P. W. Purdom, Jr. Search Rearrangement Backtracking and Polynomial Average Time. *Artificial Intelligence,* 21:117-133, 1983.

[Sacerdoti, 1975] E. D. Sacerdoti. The Non-linear Nature of Plans. In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 206-214, 1975.

[Stefik, 1981] M. Stefik. Planning with Constraints, (MOLGEN: part 1). *Artificial Intelligence,* 16:111-139, 1981.

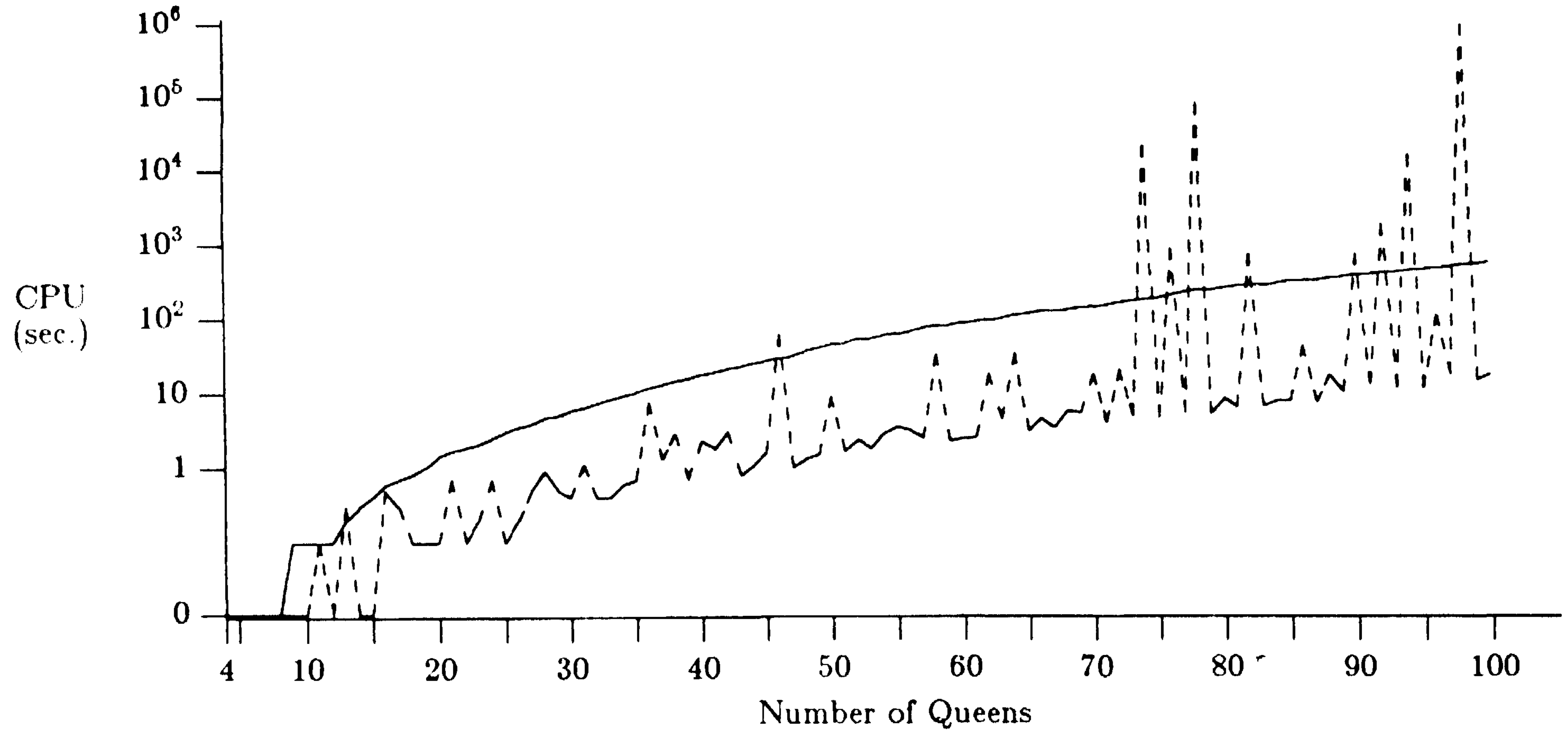
Tate, 1977] A. Tate. Generating Project Networks. In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 888-893, 1977.

Tsai, 1988] R. L. Tsai. Static Task Assignment in Parallel Processing. Dissertation, Southern Methodist University, 1988.



o -- our approach x -- simple heuristic method

Figure 2(a)



solid line — our approach dashed line — simple heuristic method

Figure 2(b)