Predicate-Transition Net Reachability Testing Using Heuristic Search

Nonmember Jaegeol Yim (Dong-Kook University at Kyung-Ju)
Nonmember Peter C. Nelson (University of Illinois at Chicago)
Nonmember Tadao Murata (University of Illinois at Chicago)

Summary

Given two markings $M_0$ and $M_g$ on a predicate-transition (pr/t) net, finding a firing sequence which transforms $M_0$ to $M_g$ is called the reachability test problem. Once a problem in the real world has been modeled as a pr/t net, a solution of the problem can be, in most cases, directly obtained from performing a reachability test on the pr/t net model. However, as a problem with exponential complexity, there does not exist an efficient algorithm to perform the reachability test. One method which might be used in solving such a problem is heuristic search. The efficiency and effectiveness of a heuristic search algorithm is dependent upon its heuristic function. We have found several admissible and monotonic heuristic functions which can be used to solve the reachability test problem. We have implemented an iterative deepening $A^*$ algorithm using our heuristic functions, and present the experimental results for our heuristics.

Key words: Petri net, Reachability, Robot planning, Heuristic search, $A^*$ algorithm

1. Introduction

The predicate/transition (pr/t) net was first defined by Genrich (1) and can be conveniently used in modeling first-order predicate logic. A sentence written in first-order predicate logic can always be translated into a set of clauses. Most of the automated reasoning algorithms (10) work on a database consisting of clauses. Therefore, Murata et al. (8) defined a (pr/t) net which is a subset of Genrich’s for modeling a logic program written in clausal form. Pr/t nets have also been used in modeling robot planning systems (5), (6), and rule-based deduction systems (7). Applying inference rules in rule-based deduction systems, and applying an action to search for a plan in a robot planning system can be simulated by firing transitions on a pr/t net model. Deriving answers in those systems becomes answering whether a certain marking, the goal marking, is reachable from the initial marking in a pr/t net model. This problem is of exponential complexity, and even though the reachability test is essential in pr/t net analysis, there are no known efficient reachability test algorithms available.

The reachability test is in fact a graph search. Unfortunately the cost of solving this problem is exponentially expensive. AI search techniques employ the use of heuristics in order to improve the efficiency of a search process. Heuristics are used to guide the search process through the state space graph in a best-first manner by exploring the most promising states first. A heuristic function based on domain knowledge is used to estimate the distance from unexplored states to the goal state. The state which appears to be closest to the goal is the next state that will be explored. We have found several admissible and monotonic heuristic functions which can be used to solve the reachability test problem. We have implemented an iterative deepening $A^*$ algorithm using our heuristic functions, and present the experimental results for our heuristics.

To economize space, it is assumed that the reader is familiar with basic concepts of Petri nets (4). The necessary understanding of the pr/t net model presented in this paper can be obtained from our previous paper (6). We briefly review heuristic search techniques in Section 2. Section 3 provides several heuristic functions, and the test results are discussed in Section 4. Finally concluding remarks are presented in Section 5.
2. Heuristic Search Techniques

The goal of our search algorithm is to find a firing sequence $a$ which transforms a given initial marking $M_i$ to a given goal marking $M_g$. Each transition has an associated cost. We assume that the firing cost is one for every transition. The cost of a firing sequence $a$ is then the number of transitions fired in $a$. The main contribution of our work is to provide admissible and monotonic heuristic functions. Using these functions, we have implemented an iterative deepening $A^*$ (IDA*) algorithm (3).

The most primitive search algorithms are breadth-first search and depth-first search. The advantage of breadth-first search is that it always finds an optimal solution if one exists, whereas the advantage of depth-first search is that it only needs $O(d)$ space, where $d$ is the depth of the search space. The disadvantage of breadth-first search is that it needs $O(bd)$ space, whereas the disadvantage of depth-first search is that it does not guarantee to find an optimal solution.

A search algorithm which possesses the advantages of both breadth-first search and depth-first search is the depth-first-iterative-deepening (DFID) search algorithm. It repeatedly performs a depth-first search of depth $i$ where $i$ varies from 1 to infinity until a solution is found. As an illustrative example of DFID algorithm, consider the search space shown in Figure 1. DFID traverses Node 1 when $i$ is 1, or for the first iteration. When $i$ is 2, or for the second iteration, DFID traverses Node 1, Node 2 and Node 3 in that order. When $i$ is 3, DFID traverses Node 1, Node 2, Node 4, Node 5, Node 3, Node 6 and Node 7 in that order. The search process continues until the goal state is reached.

There have been numerous heuristic search techniques developed (9). Perhaps the most well-known heuristic algorithm is $A^*$ developed by Hart et al (2). The $A^*$ algorithm orders nodes for exploration using a heuristic function $f = g + h$, where $g$ is the known distance from the initial state to the current state and $h$ is an estimate of the remaining distance from the current state to the goal. The $A^*$ algorithm is a best-first search since it selects the most promising node with the minimum $f$ value among the unexplored nodes. A heuristic function is admissible if it never overestimates. If $A^*$ is used with an admissible heuristic, it will find an optimal solution if one exists. Similar to breadth-first search, the disadvantage of $A^*$ search is that it needs $O(b^d)$ space in the worst case. The combination of the $A^*$ algorithm and DFID is the iterative-deepening $A^*$ (IDA*) algorithm (3). Similar to DFID, IDA* continually performs a depth-first search of depth $i$ where $i$ varies from 1 to some limit depth. The difference is that IDA* discards a node with an $f$ value which exceeds a given threshold. The threshold is initialized to the $f$ value of the initial state, and threshold used for the next iteration is the minimum $f$ that exceeded the current threshold. It is known (3) that if the heuristic used by IDA* is admissible, then IDA* will find an optimal solution if one exists.

A heuristic function $h$ is said to be monotonic if
\[ c(M_i, M_j) + h(M_j) \geq h(M_i), \]
where $M_j$ is a successor of $M_i$, $c(M_i, M_j)$ is the actual cost of an optimal path from $M_i$ to $M_j$, and $h(M_i)$ and $h(M_j)$ are the estimated costs of optimal paths from $M_i$ to $M_j$ and $M_j$ to $M_g$, respectively. Using the $A^*$ algorithm with a monotonic heuristic function is simpler than with a non-monotonic heuristic function as nodes will never be re-expanded.

3. Heuristic functions

The efficiency of the IDA* algorithm depends on its heuristic function. We have found four possible heuristic functions which can be used in solving the reachability test.

3.1. Heuristic $h_m$

Suppose the task assigned to a robot is to fetch box A and box B. Let $a_1$ denote the cost of bringing box A, and $b_1$ denote the cost of bringing box B. The cost to fulfill the task of fetching both boxes A and B is obviously not less than $a_1 + b_1$. Thus, Maximum($a_1 + b_1$) can be used as an underestimate of the cost to fulfill the whole task. Similarly, given a petri net with a goal marking $M_g$ consisting of goal tokens $g_1, g_2, \ldots$ in certain places, the cost to achieve $M_g$ will not be less than the cost of achieving the most expensive goal token. Let the optimal cost to achieve $g_1$ be denoted by $\sigma[g_1]$, which is the minimum number of transitions that must be fired to produce $g_1$. Then the maximum value among $\sigma[g_1], \ldots, \sigma[g_n]$ can be used as an underestimate of the cost to achieve the goal $M_g$. Our first heuristic function $h_m$ is to calculate the maximum value among $\sigma[g_1], \ldots, \sigma[g_n]$.

\[ h_m(M_i, M_g) = \text{Max}(\sigma[g_1], \sigma[g_2], \ldots, \sigma[g_n]), \]
where $M_i$ is the current marking, and $n$ is the number of tokens in $M_g$.

**Theorem 1:** Heuristic function $h_m$ is admissible, i.e.,
\[ h_m(M_i, M_g) \leq c(M_i, M_g), \]
where $c(M_i, M_g)$ is the cost of the optimal firing sequence $a$ by which $M_g$ is reachable from $M_i$.

**Proof:** This follows from the fact that $\sigma[a] \geq a_1, a_2, \ldots, \gamma_n$. 

Theorem 2. Heuristic function $h_m$ is monotonic, i.e.,
$$c(M_i, M_j) + h_m(M_j) \geq h_m(M_i).$$

Proof by induction: basis step: Let $M_j$ be reachable from $M_0$ by firing a transition $t_1$, i.e., $M_0[t_1] = M_j$. Then $c(M_0, M_j) + h_m(M_j) \geq h_m(M_0)$ because of the following reason: By the definition, $h_m(M_0) = \text{Max}(\sigma_1, \sigma_2, ..., \sigma_n)$ and $h_m(M_j) = \text{Max}(\sigma_j, \sigma_2, ..., \sigma_n)$, where $\sigma_j$ is the optimal cost to achieve $s_j$ from $M_j$. Since $M_0[t_1] = M_j$, $\sigma_j \leq \sigma_j + 1$ for every $i$, $1 \leq i \leq n$. Therefore, $1 + h_m(M_j) \geq h_m(M_0)$.

induction hypothesis: Let $M_k[t_i] = M_j$, then $c(M_k, M_j) + h_m(M_j) \geq h_m(M_k)$. Applying the same argument as we used for the basis step to $M_k, M_j$ and $t_i$ instead of $M_0, M_i$ and $t_1$, respectively, we can conclude
$$c(M_i, M_j) + h_m(M_j) \geq h_m(M_i).$$

By combining this result with induction hypothesis, we can conclude
$$c(M_i, M_j) + h_m(M_j) \geq h_m(M_i).$$

The fewer the number of tokens in $M_g$, the closer $h_m$ is to $c$. If a goal marking has only one goal token, then our estimate is identical to the real cost of an optimal solution. In order to see how powerful the heuristic function $h_m$ is, let us solve Sussman’s anomalous problem (11) using the A* algorithm with $h_m$. Sussman’s anomalous problem occurs in the well-known Blocks World robot planning problem. In the Blocks World robot planning problem, there are several blocks on a table. The configuration of the blocks at any given time is called a state. A robot hand has the ability to move the blocks around. It can pick up a block if it is on the table and its top is clear. It can also unstack a block from another block. This problem can be modeled in a prf net as shown in Figure 2 (6). The object is to find a sequence of actions, or a plan, which if performed by the robot hand will change the configuration of the blocks into the desired state.

When given the initial state and the goal state as shown in Figure 3, finding a plan is called Sussman’s anomalous problem. This problem is called an anomalous problem because of the following reason. In order to achieve the goal of this problem, the robot has to stack B on C and A on B. Stacking A on B cannot be done before stacking B on C, because in order for B to be stacked on C, B must have a clear top. If we apply a stacking B on C operation to the initial state, then the state is changed into a state, where stacking A on B becomes more difficult. The famous robot planning system STRIPS failed in solving this problem. This problem is equivalent to finding an optimal firing sequence to reach $M_g = \{D, D, D, D, [\langle A, B \rangle, [\langle B, C \rangle], [\langle A, B \rangle], [\langle B, C \rangle]], [\langle C, A \rangle]\}$, where D stands for a "don’t care" condition, and the order of places is ONTABLE, CLEAR, HANDEMPTY, HOLDING and ON, on the prf net shown in Figure 2.

Figure 2. The prf net model of the Blocks World.

Fig. 3. Sussman’s anomalous Blocks World robot planning problem

Example: Using the A* algorithm with $h_m$ to solve Sussman’s anomalous problem, the search would proceed as shown in Figure 4. At the beginning there is only one node, node 1, in the search tree. After expanding node 1, the search tree consists of nodes 1, 2, and 3. The names of nodes are labeled on the upper left corner of each node. The estimated cost of optimal paths from $M_0$ to $M_g$ through each node is labeled in the upper right corner of each node. The estimated cost for node 3 is 1+3. Therefore, A* selects node 3 to expand next. After expanding node 3, the search tree consists of nodes 1, 2, 3, 4, 5, and 6. The node with the minimum estimate is node 4. After expanding node 4, there are six open (leaf) nodes, node 2, 6, 7, 8, and 9, in the tree. Their estimates happen to all be the same. When there is more than one node with a minimum estimate, we arbitrarily select the deepest node. At this moment node 2 is at depth 1, nodes 5 and 6 are at depth 2, and nodes 7, 8, and 9 are at depth 3. When there is more than one node at the same depth with the same minimum estimate, we select one of them arbitrarily. Suppose we select node 5. After expanding node 5, nodes 2, 6, 7, 8, 9, and 11 are open nodes. Note, node 12 is identical to node 4. We may want to check for duplicate nodes and delete it or we may want to save time by not checking for duplicate nodes. If we do not check for duplicate nodes, A* would select node 12 and expand it. 8 is the f value of every child node of node 12. In any case, A* eventually selects node 8 whose f value is 6. The
algorithm continues until finding the goal as shown in the figure.

Figure 4. A search tree constructed by A" algorithm using htn.

For this example, the number of explored nodes is very close to the minimum number of nodes which must be explored. We made two arbitrary decisions in the example. One was selecting the deepest node when there is more than one node with a minimum estimate. This is a good policy, if we have confidence in our heuristic function. We assume that our heuristic function is good, and whenever we apply an action we assume that the state becomes closer to the goal state.

The second arbitrary decision was made in selecting node 7 first out of nodes 7, 8, and 9. Node 8 was the correct choice. In the example, we first made a wrong decision of selecting node 7. Then we selected node 8. Therefore the search space shown in Figure 4 is an average case. The problem with htn is that it is not available. Computing htn is as hard as computing m itself. Therefore we have to come up with a new heuristic function which is easier to compute.

3.2. Heuristic h0

Although htn is a powerful heuristic function, computing htn is so time consuming that we cannot make use of it in practice. As a heuristic function with zero knowledge, we define h0 which always returns 0. Heuristic h0 is obviously admissible and monotonic. Since h0 always returns 0, IDA* using h0 becomes a DFID search.

3.3. Heuristic h1

Computing h0 is very simple but unfortunately it does not have any heuristic power. As an improved heuristic function, we define h1. Given the goal marking, Mg, and the current marking Mi, h1 first computes the difference between Mg and Mi. The difference between Mg and Mi is denoted by Mdiff = Mg - Mi, and is defined by the sets of tokens which appear in Mg but not in Mi. The heuristic value, h1(Mi, Mdiff) is defined to be 0 if there exists a transition which has output places to all the corresponding non-empty elements of Mdiff; otherwise it is 1.

For example, consider the small pr/t net shown in Figure 5(a). The current marking of the net is M1 = {{<A>}, {2}, {1}}. Let Mg = {{}, {<A>}, {<A>}, {<A>}}. Then, h1 first computes Mdiff = {{}, {<A>}, {<A>}, {<A>}}. Since the set of output places of t includes the set of all the places corresponding to the non-empty elements of Mdiff, h1 returns 0.

Now, consider the pr/t net shown in Figure 5(b). Let Mg = {{}, {<A>}, {<A>}}. Since there is no proper unifier, Mg cannot be achieved by one transition firing. However, h1 still returns 0 because of the same reason for the case of the pr/t net shown in Figure 5(a). This happens because we ignore unifiers in computing h1.

This function is obviously admissible. This function is also monotonic because the distance between Mi and Mg cannot be decreased by more than 1 for any action applied to Mg and the value of h1 is not greater than 1. This function is very simple and if the difference between Mi and Mg is great, as is usually the case at the beginning of a search, then the h1 value would be 1.

Figure 5. Two small pr/t nets.

3.4. Heuristic h2

Given markings Mi and Mg, consisting of tokens g1, g2, ..., let d*(Mi, gi) be the approximate distance from Mi to gi, where gi is a token which appears in Mg. Ideally, d*(Mi, gi) should be the minimum number of transitions to be fired from Mi to produce gi.

However, computing the exact number of transitions in an optimal path is very time consuming. Therefore we define:

d*(Mi, gi) = the number of transitions to be fired from Mi to produce gi ignoring conflicts between transitions.

Figure 6. An example place/transition net for explanation of h2.

For example, consider the place/transition net shown in Figure 6. Let M1 = (1, 0, 1, 1, 0, 0) and Mg = (0, 0, 0, 0, 0, 11). In order to...
reach $M_g$, we have to fire $t_4$, $t_1$, $t_2$, and $t_3$ in that sequence. Therefore actual distance from $M_1$ to $M_g$ is 4. If we ignore interactions between transition firings, that is tokens are never consumed, the approximate distance between $M_1$ and $M_g$ is 2, firing $t_1$ and $t_3$.

Before defining $h_2'$, we need to define $h_2$ as:

$$h_2(M_1, M_2) = \text{Max}(d^* (M_1, g_1), d^* (M_1, g_2), \ldots, d^* (M_1, g_n)),$$

where $n$ is the number of tokens appear in $M_g$. Implementing this function for a prh net is much more complicated than the impression readers might have from this example. Evaluation of $h_2'$ involves computing subgoals for every goal token. Computing a subgoal requires considering a unifier. Thus, $h_2'$ is a very time consuming function.

**Theorem 3:** Heuristic $h_2'$ is admissible.

**Proof:** This follows from the fact $d^* (M_1, g_1) \leq \text{deg}_1$.

**Theorem 4:** Heuristic $h_2'$ is monotonic.

**Proof by induction:** Exactly the same as the proof of Theorem 2, except $h_0$ is replaced with $h_2'$ and $\sigma$ is replaced with $d^*$.

By combining $h_1$ with $h_2'$, we define $h_3$ as:

$$h_3(M_1, M_2) = h_1(M_1, M_2) + h_2'(M_1, M_2).$$

Recall that if $h_1$ is 1 then the goal marking cannot be reached by firing one transition, and $d^*$ is approximation of the cost to produce one goal token. Therefore, $h_3$ is a very time consuming function.

**3.5. Heuristic $h_3$**

So far we have discussed general purpose heuristic functions which could be applied to any prh net. For a particular problem, we may find a fast and powerful heuristic function. For example, consider the prh net shown in Figure 2 which is a model of the Blocks World problem shown in Figure 3. Since there is only one robot hand working, the robot cannot put down A and B at the same time to achieve O N T A B L E (A) and O N T A B L E (B). Similarly the robot cannot stack A on B and B on C at the same time to achieve ON(A, B) and ON(B, C). This is equivalent to saying that we cannot add more than one token to a single place by firing one transition. Thus, the maximum number of atomic formulas of a single predicate appearing in the difference between the goal state and the current state can be used as the underestimate of the distance between the states. This can be stated in terms of prh net model as follows: Given $M_1$ and $M_g$, compute $M_g = M_g - M_1$. For every place, let the number of corresponding tokens appearing in $M_g$ be denoted by $c_1, c_2, \ldots$ Let $\text{Max}(c_1, c_2, \ldots)$ be denoted by $c$. Then, $c$ is an underestimate of the number of transitions to be fired to achieve the goal marking, $M_g$.

Furthermore, consider the number of actions required to satisfy two formulas of the same predicate, for example, O N T A B L E (A) and O N T A B L E (B). In order to achieve formula O N T A B L E (A) or O N T A B L E (B), the robot must first hold A or B. But the robot cannot hold both A and B at the same time. The optimal state to reduce the difference containing O N T A B L E (A) and O N T A B L E (B) is a state in which one of the blocks is being held by the robot while the other block has a clear top so that the robot can pick it up after put down the block being held. Thus, the minimum number of actions which must be performed by the robot to reduce the difference O N T A B L E (A) and O N T A B L E (B) is at least three. If we extend the number of formulas of the same predicate to three, for example O N T A B L E (A), O N T A B L E (B) and O N T A B L E (C), then the minimum number of actions which must be performed by the robot becomes five. In general, let the maximum number of formulas of a single predicate appears in the difference between the goal state and the current state be denoted by $c$. Then, the minimum number of actions which must be performed by the robot for the Blocks World problem is $(c+2) - 1$. Thus, heuristic function $h_3$ is defined as: $h_3(M_1, M_2) = (c+2) - 1$.

Applying the similar arguments which have been used in proving that the previous heuristics are admissible, we can easily prove that $h_3$ is admissible. However, $h_3$ is not monotonic due to multiplication, $(c+2)$.

4. Test Results

In order to verify the heuristic power, we have implemented IDA* using heuristics $h_0$, $h_1$, $h_2$, and $h_3$. Our experiments were done on the prh net shown in Figure 2. The important factors for the size of search space are the length of the solution and the branching factor. The branching factor for this problem depends on the number of blocks and the configuration of the blocks. In order to guarantee that the branching factor increases as the number of blocks increases, we put all the blocks on the table for the initial state of our test input problem. A typical problem for our test run is shown in Figure 7. It is a 5 blocks world problem and the length of solution is 8. The initial marking corresponding to the initial state is

$$M_1 = \{ [A], [B], [C], [D], [E], \}.$$

and the goal marking is

$$M_g = \{ [A, B], [C, D, E], \}.$$

![Initial state](image1)

![Goal state](image2)

Figure 7. A typical sample case.
The relationship between the length of the solution and the number of nodes explored is shown in Figure 8. Using IDA* with \(h_0\) is really just DFID. As the length of solution increases, the discrepancies between \(h_0, h_1, h_2,\) and \(h_3\) become more and more significant. The discrepancies between them also become more significant as the branching factor increases. For the 4 Blocks World problem (Figure 8a), the number of nodes explored by \(h_2\) is less than half but greater than one third of the number of nodes explored by \(h_0\) when the solution length is 6. For the 6 Blocks World problem (Figure 8d), the number of nodes explored by \(h_2\) is less than one fourth of the number of nodes explored by \(h_0\). The number of nodes explored by \(h_2\) and \(h_3\) is always much less than the number of nodes explored by \(h_0\) and \(h_1\). The discrepancy between \(h_2\) and \(h_3\) seems insignificant in the graph, but we can expect that it would become more significant if the length of solution becomes longer. The curve fits are as follows:

\[h_0: \quad y = 0.65 \times 10^{0.72x}\]
\[h_1: \quad y = 0.73 \times 10^{0.68x}\]
\[h_2: \quad y = 0.45 \times 10^{0.63x}\]
\[h_3: \quad y = 0.56 \times 10^{0.62x}\]

As expected, all the curves are exponential. However we also see that heuristics \(h_1, h_2,\) and \(h_3\) provided an exponential reduction in the number of nodes expanded with respect to the length of solution.

Figure 8e shows the relationship between the number of nodes explored and the branching factor by combining the 4 different graphs shown in Figure 8a,b,c,d. For our input data, it is guaranteed that the branching factor increases as the number of blocks does. We see that as the branching factor increases, the savings provided by the heuristics also increases. The numbers of nodes explored by \(h_2\) and \(h_3\) are always less than the numbers of nodes explored by \(h_0\) and \(h_1\). Again the reduction in the number of node expansion by \(h_1, h_2,\) and \(h_3\) over \(h_0\) is exponential with respect to the number of blocks.

We mentioned that the heuristic function \(h_2\) for prf nets is a very time consuming function whereas \(h_1\) and \(h_3\) are quite fast. The relationship between CPU time and the length of solution is shown in Figure 8f. Even though the number of nodes explored by \(h_2\) is much less than the number of nodes explored by \(h_0\) and \(h_1\), the amount of CPU time spent by \(h_2\) is much greater than the amount of time spent by \(h_1\) and \(h_3\). In fact, \(h_2\) spends more time than \(h_0\) which is our zero knowledge heuristic. This result confirms that our heuristic function must be simple to calculate.

5. Conclusion

The predicate transition net reachability test problem is of exponential complexity. Heuristic search can be used in solving such problems of exponential complexity. The efficiency of a heuristic search depends on its heuristic function. We have discussed several admissible and monotonic heuristic functions. These functions have been tested using the IDA* algorithm. The test results show that we can reduce both CPU time and the number of nodes explored by using a heuristic function.

Our current research includes developing better heuristic functions. This may be able to be achieved by combining existing heuristics, or possibly through the use of specialized heuristics for certain problem domains.

(Manuscript received March 15, 1994)
References


Jaegul Yim is an Assistant Professor in the Department of Computer Science at the Dong-Kook University at Kyung-Ju in Korea. His research interests include application of Petri net theory to robot planning and logic programming, applied graph theory, and computer science education. He received his B.A. degree from the Dong-Kook University in 1982 and his M.S. and Ph.D. degrees in E.E.C.S. from the University of Illinois at Chicago, in 1987 and 1990, respectively.

Peter C. Nelson is an Associate Professor in the Department of Electrical Engineering and Computer Science at the University of Illinois at Chicago. He received his Ph.D. degree from Northwestern University in 1988. His current research interests include heuristic search, intelligent vehicle highway systems, automated restriction mapping for the Human Genome Project, and computer-aided manufacturing.

Tadahiro Murata received the M.S. and Ph.D. degrees in Electrical Engineering from the University of Illinois at Urbana-Champaign, in 1964 and 1966, respectively. He is presently a Professor in the Department of Electrical Engineering and Computer Science at the University of Illinois at Chicago, where he initially joined in 1966. During occasional leaves of absence from the University of Illinois, he taught at the University of California at Berkeley, Tokai University, Tokyo, Japan, and Osaka University (as Endowed Hitachi-Chair Professor for 1993-94), Toyonaka, Japan. Also, he was invited to visit Dr. C. A. Petri's Institute at GMD, mbH, Germany, and several other research institutes and universities in Europe. In 1990, he received a distinguished faculty award from the President of the University of Illinois. He is the recipient of the 1991 IEEE Donald G. Fink Prize Award.

His current research interests include Petri net theory and its applications, especially to the modeling and analysis of concurrent, parallel and/or distributed processing systems, logic programming and AI applications. In these areas, he has published extensively and been awarded several NSF grants since 1976. Prior to that he worked in the area of circuits, systems, and applied graph theory.

Dr. Murata has served on the U.S. National Academy of Science/Computer Science and Technology Board. He served as the general chairman for the 1987 International Workshop on Petri Nets and Performance Models, Madison, WI, and organized the 1993 International Conf. on Application and Theory of Petri Nets, Chicago, IL. He has served on Editorial Board of the *IEEE Transactions on Software Engineering*, and is currently serving as an associate editor for *Journal of Circuits, Systems and Computers*.

He is a fellow of the IEEE, a member of European Association of Theoretical Computer Science, Information Processing Society of Japan, and Special Interest Group on Petri Nets and Related Systems Models, Gesellschaft für Informatik, FRG. He is listed "Who's Who in America," among others.