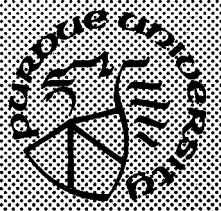
7-1-1987

Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System

C. L. Chen
Purdue University

C. S. G. Lee
Purdue University

S. H. E. Hou
Purdue University



Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System

C. L. Chen
C. S. G. Lee
S. H. E. Hou

TR-EE 87-27
July 1987

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System

C. L. Chen, C. S. G. Lee, and S. H. E. Hou

Ransburg Robotics Laboratory

School of Electrical Engineering

Purdue University

West Lafayette, Indiana 47907

TR-EE 87-27

July 1987

Abstract

Robot manipulators are highly nonlinear systems and their motion control requires the computation of generalized forces/torques to drive all the joint motors at an adequate rate. This paper presents efficient scheduling algorithms for computing the robot inverse dynamics on a multiprocessor system. The problem of scheduling the inverse dynamics computation consisting of m computational modules to be executed on a multiprocessor system consisting of p identical homogeneous processors to achieve a minimum-scheduled length is known to be *NP*-complete. In order to achieve the minimum computation time, the Newton-Euler equations of motion are expressed in the homogeneous linear recurrence form which results in achieving maximum parallelism. To speed up the searching for a solution, a heuristic search algorithm called Dynamical Highest Level First/Most Immediate Successors First (*DHLF/MISF*) is first proposed to find a fast but suboptimal schedule. For an optimal schedule, the minimum-scheduled-length problem can be solved by a state-space search method — the A^* algorithm coupled with an efficient heuristic function derived from the Fernandez and Bussell bound. The state-space search method is a classical minimum cost graph search algorithm, which is guaranteed to find the optimal solution if the evaluation function is properly defined. An objective function is defined in terms of the task execution time and the optimization of the objective function is based on the minimax of the execution time. The proposed optimization algorithm solves the minimum-scheduled-length problem in pseudo-polynominal time and can be used to solve various large-scale problems in a reasonable time. An illustrative example of computing the inverse dynamics of an n-link manipulator based on the Newton-Euler dynamic equations is performed to show the effectiveness of the A^* algorithm and the heuristic algorithm *DHLF/MISF*.

This work was supported in part by the NSF-Purdue Engineering Research Center under Grant CDR-8500022.

1. Introduction

Robot manipulators are highly nonlinear systems and their motion control involves the computation of the required generalized forces/torques, from an appropriate manipulator dynamics model, using the measured data of displacements and velocities of all the joints, and the accelerations computed from some justifiable formulae or approximations, to drive all the joint motors. Obviously, the execution time for computing the generalized forces partially determines the feasibility of implementing the control scheme in real time. There are a number of ways to compute the applied generalized forces/torques, among which the computation of joint torques from the Newton-Euler (NE) equations of motion is the most efficient and has been shown to possess the time lower bound of $O(n)$ running in uniprocessor computers [1,2], where n is the number of degrees of freedom of the manipulator. It is unlikely that further substantial improvements in computational efficiency can be achieved, since the recursive NE equations are efficiently computing the minimum information needed to compute the generalized forces/torques: angular velocity, linear and angular acceleration, and joint forces and torques. Nevertheless, some improvements could be achieved by taking advantage of particular computation structures [3], customized algorithms/architectures for specific manipulators [4,5], parallel computations [6,7], and scheduling algorithms for multiprocessor systems [8-10].

The approach of particular computation structures requires the reformulation or manipulation of equations of motion to optimize the speed of the architectures, while customized algorithms/architectures are designed to improve the computational efficiency by taking advantage of particular kinematic and dynamic structures of the manipulator. Parallel computations require the NE equations of motion to be expressed in a homogeneous linear recurrence form and the processors are connected efficiently to reduce the communication and buffering problems. This approach was found to be very efficient for computing the NE equations of motion [7], but may not be very efficient for computing other robotic computational tasks such as the inverse Jacobian computation. On the other hand, efficient scheduling algorithms can be used to schedule a computational task, expressed in a directed task graph, to be executed parallelly in a set of connected processors to achieve a minimum computation time. This approach does not require the computational task to be expressed in any specific mathematical form. Furthermore, the directed task graph preserves the precedence relations among the computational modules of the task.

This paper presents efficient scheduling algorithms on a multiprocessor system which determine an optimal and/or suboptimal schedule for computing the inverse dynamics of an n-link manipulators (with prismatic/rotary joints) in minimum time. The NE equations of motion are expressed in a homogeneous linear recurrence form and decomposed into m computational modules which are scheduled to be executed on p identical homogeneous processors to achieve a minimum computation time. In order to achieve the minimum computation time, it is desirable to execute many independent

modules simultaneously to achieve maximum parallelism. It has been shown in the literatures [11,12] that if the number of processors is more than two, the processing time of each module is of equal length, and the precedence relation of the modules is arbitrary, then the complexity of finding the optimal schedule to achieve a minimum-scheduled length is *NP*-complete. Furthermore, if the execution time of all the modules is arbitrary, then the optimal solution of the scheduling problem becomes a strong *NP*-complete. Thus, the problem of scheduling the inverse dynamics computation on a multiprocessor system is extremely difficult to solve and generally intractable.

Several approaches to the general multiprocessor scheduling problem have been proposed [13-17]. They are basically graph theoretical, integer programming, and heuristic methods. The graph theoretical approach uses a graph to represent application tasks. The integer programming method is based on the implicit enumeration algorithm subject to the task constraints. The heuristic methods are to provide fast and efficient algorithms for a suboptimal solution. In computing the inverse dynamics for a Stanford arm, Luh and Lin [8] assigned one microprocessor to each manipulator link and proposed a variable branch-and-bound search algorithm to find a subtask-ordered schedule for the microprocessors to compute the joint torques using the NE equations of motion. With this computational structure, the authors reported a concurrency factor of 2.64 on a Stanford arm. Kasahara and Narita [9,17] proposed a depth-first/implicit heuristic search method, which combines the branch-and-bound method and the critical path method, to compute the inverse dynamics of a Stanford arm. Barhen proposed a ROSES algorithm [10], which uses heuristic techniques with special instance of abstract data structures, to run on a hypercube machine to compute the inverse dynamics of the same Stanford arm. All of the above scheduling algorithms focus on a specific robot arm, the Stanford arm, and are difficult to generalize to other robot manipulators.

In this paper, we first propose an efficient heuristic algorithm, called Dynamical Highest Level First/Most Immediate Successors First (*DHLF/MISF*), to obtain a fast but suboptimal solution for scheduling the p processors in a multiprocessor system to compute the inverse dynamics of an n-link manipulator. Next, the well-known A^* search algorithm [18] coupled with an efficient heuristic function derived from the Fernandez and Bussell bound is proposed to obtain the optimal schedule for the robot inverse dynamics computation. The A^* search algorithm is a classical minimum cost graph search algorithm. It is guaranteed to find an optimal solution if the evaluation function which utilizes the heuristic information about the problem for speeding up the search is properly defined. Alternately, the A^* algorithm can be considered to be a branch-and-bound search using the dynamic programming principle with a cost estimate of remaining unassigned modules [18]. If the cost estimate of the remaining unassigned modules is a lower bound estimate of the actual cost, then the A^* search algorithm will produce an optimal solution. An objective function is defined in terms of the task execution time and the optimization of the objective function is based on the minimax of the execution time. The proposed optimization algorithm solves the

minimum-scheduled-length problem in pseudo-polynominal time, and based on our computer simulation, the algorithms can solve various large-scale problems in a reasonable time. An illustrative example of computing the inverse dynamics of an n-link manipulator (with prismatic/rotary joints) is performed to show the effectiveness of the A^* algorithm and the heuristic algorithm *DHLF/MISF*.

2. Maximum Parallelism of Newton-Euler Task Graph

The problem of computing manipulator joint torques based on a manipulator dynamic model is often referred to as the *inverse dynamics* problem and can be stated as: Given the joint positions and velocities $\{q_j(t), \dot{q}_j(t)\}_{j=1}^n$ which describe the state of an n-link manipulator at time t, together with the joint accelerations $\{\ddot{q}_j(t)\}_{j=1}^n$ which are desired at that time, solve the dynamic equations of motion for the joint torques $\{\tau_j(t)\}_{j=1}^n$ as follow:

$$\tau(t) = \mathbf{f}(\mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)) \tag{1}$$

where

$$\tau(t) = (\tau_1, \tau_2, \cdots, \tau_n)^T, \ \mathbf{q}(t) = (q_1, q_2, \cdots, q_n)^T,$$
$$\dot{\mathbf{q}}(t) = (\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_n)^T, \ \ddot{\mathbf{q}}(t) = (\ddot{q}_1, \ddot{q}_2, \cdots, \ddot{q}_n)^T,$$

the superscript T denotes transpose operation on vectors and matrices, and Eq. (1) indicates the functional representation of the manipulator dynamic model. Since the NE equations of motion have been known for their efficiency in computing the joint torques whether they are formulated in the base coordinate system [2] or in the link coordinate systems [1], our objective is to see how fast one can schedule the computation of the NE equations of motion on a multiprocessor system with p identical processors to achieve a minimum computation time.

In general, a computational task can be represented by a directed acyclic task graph (DATG) $G = (V, E)$ consisting of a finite nonempty set of vertices V, $V = (T_1, T_2, \cdots T_m)$, and a set of finite edges E, $E = (e_1, e_2, \cdots)$, connecting them. Each vertex represents a computational module (CM) and each edge represents a precedence constraint between two CMs. An edge connecting module T_i to module T_j is denoted by $e(i, j)$. The precedence constraint between CMs indicates which modules have to be completed before some other modules can be started. Our optimal scheduling problem is to assign these modules of a DATG to the p processors so that the precedence relation is not violated and that all the modules together are processed in the shortest possible time (i.e. in a minimum computation time). The time that the last module in a schedule is completed is called the finishing time of the schedule. Thus, we want to minimize the finishing time of a given DATG over all the permissible schedules.

In order to schedule the computation of the NE equations of motion over a set of p processors, we need to represent or express the NE equations of motion as a DATG with

its vertices indicating the computational modules and its edges indicating the precedence relation among the modules. An examination of the recursive NE equations of motion shows a certain amount of parallelism with a large amount of sequentialism in the flow of computation. This serial nature of the computational flow lends itself to a pipelined implementation [4]. In order to achieve parallel processing with minimum computation time, it is desirable to develop a directed task graph with maximum parallelism for the NE equations of motion. Unfortunately, there are no general procedures in generating a maximum-parallelism task graph from the NE equations. Intuitively, one can decompose the NE equations into elementary operations such as multiplication/division, addition/subtraction, and trigonometric functions. A better approach is to perform a functional decomposition of the NE equations; that is, the equations are decomposed into computational modules, each of which calculates the kinematic and dynamic variables such as angular velocities, angular and linear accelerations, joint forces and moments, etc. This "macro-decomposition" results in a maximum-parallelism task graph with its computational modules corresponding to the terms used to generate the recursive forward and backward equations. This macro-decomposition also allows us to obtain a maximum-parallelism task graph for any n-link manipulator with prismatic/rotary joints. Since the NE equations of motion can be formulated either in the base coordinate system or in the link coordinate systems, the task graphs obtained by the macro-decomposition technique for these two formulations are different. The clear advantage of referencing both the kinematic and dynamic variables to the link coordinates is to obviate a great deal of coordinate transformations and to allow the inertia tensor to be fixed in each link coordinate frame, which results in a much faster computation in a uniprocessor computer. However, the recursive structure of this formulation is found to be in an inhomogeneous linear recurrence form (IHLR) which is not efficient for parallel processing [7]. On the other hand, when expressed in the base coordinate system, the NE equations are in a homogeneous linear recurrence form (HLR) which is more suitable for parallel processing [7]. In either form, the use of the recursive doubling technique for the parallel computation of the NE equations results in a time order of $O(\log_2 n)$. The linear recurrence structure of the NE equations expressed as a DATG is shown in Fig. 1.

For a linear recurrence equation,

$$x_i = a_i x_{i-1} + b_i \qquad ; \; i = 1, 2, \cdots, n \tag{2}$$

if $a_i = 1$, then it is in a HLR form; if $a_i \neq 1$, then it is in an IHLR form. If x_i and b_i are 3×1 vectors and a_i is a 3×3 matrix, and if we further assume that parallel computation on vector and matrix operations are available, that is, parallel computation of two 3×1 vector addition takes 1 add, a dot product requires 1 mult and 2 adds, a vector cross product takes 1 mult and 1 add, and a matrix-vector multiplication takes 1 mult and 2 adds, then using the recursive doubling technique, it takes $\lceil \log_2 n \rceil$ adds to evaluate Eq. (2) in the HLR form parallelly, while it takes $\lceil \log_2 n \rceil$ mults and $2 \lceil \log_2 n \rceil$ adds to

evaluate Eq. (2) in the IHLR form parallelly. Thus, the critical path of the task graph of the NE equations expressed in the HLR form (i.e. expressed in the base coordinate system) will be shorter than the one expressed in the IHLR form. Since a shorter critical path results in a shorter computation time, it is advantageous to express the NE equations with respect to the base coordinate system. The NE task graphs in the HLR form and in the IHLR form are shown, respectively, in Figs. 2 and 3. The detailed decomposition of the NE equations into task graphs in both the HLR and IHLR forms are described, respectively, in Appendices B and C, and the detailed description of the computation of each module is listed, respectively, in Tables 1 and 2. From Table 1 and Fig. 2, by counting the processing time through all the paths from the initial vertex to the terminal vertex, it is not difficult to see that the critical path of the NE task graph in the HLR form passes through modules 1-3, modules 5-6, modules 9-11, modules 15-17, module 19, and modules 27-30, with a dominant processing time of $\{3\lceil \log_2(n+1)\rceil + 4\lceil \log_2 n\rceil + (10-\lambda_i)\}$ adds and $\{\lceil \log_2 n\rceil + (6-2\lambda_i)\}$ mults, where λ_i is a joint indicator; $\lambda_i = 0$, if joint i is rotary, and $\lambda_i = 1$, if joint i is prismatic. The critical-path computation means that if the number of processors is unlimited, the above minimum computation time of the NE equations can be achieved for any n-link manipulator. Similarly, the critical path for the NE task graph in Fig. 3 passes through modules 1-3, modules 5-7, modules 11-13, modules 17-19, and modules 27-30, with a dominant processing time of $\{6\lceil \log_2(n+1)\rceil + 4\lceil \log_2 n\rceil + 8\}$ adds and $\{3\lceil \log_2(n+1)\rceil + 2\lceil \log_2 n\rceil + (6-2\lambda_i)\}$ mults, which has a longer computation time. We shall use the NE task graph in Fig. 2 to schedule the computation of the NE equations on a multiprocessor system with p identical processors. Note that each of these modules in the task graph can be further decomposed into elementary operations. Table 3 shows a comparison among various methods for the robot inverse dynamics computation.

Next, we need to define and formulate an objective function for our scheduling problem. Consider the DATG as shown in Fig. 4a. We introduce the ordered pair (T_i, D_i)† for labeling the modules, which means that module i, T_i, has a D_i unit of execution time. If there is an edge from module x to module y, then module x is said to be an immediate predecessor of y, (or equivalently module y is an immediate successor of x) and we denote it as $IPRED(y) = x$. If there is a directed path from module x to module y, then module x is said to be a predecessor of y, (or equivalently module y is a successor of x) and we denote it as $PRED(y) = x$. Initial modules are those modules with no predecessors, and terminal modules are those modules with no successors. The level l_i of a module T_i is the summation of the execution time associated with the modules in a path from T_i to a terminal module such that this sum is maximal. Such a path is called the *critical path* if the module T_i is the highest level in the DATG [1], and

† We will also alternately write T_i to represent the module i.

we define the critical-path length as

$$D_{cp} \triangleq \max_{T_i \in V} l_i \tag{3}$$

where D_{cp} is the minimum possible finishing time for the multiprocessors to process all the modules in a given DATG. The physical meaning of the *critical path* is, whatever which scheduling method is employed, the finishing time over all permissible schedules cannot be shorter than the D_{cp}. Based on the above discussion, we need to define an objective function and the optimization criterion for determining an optimal schedule which will achieve the minimum processing time for a given DATG.

For a given DATG, let $t_k(S)$ be the total computation time spent in processor k, $1 \leq k \leq p$, for a schedule S, $S \in \Omega$, where Ω is the set of all possible schedules for the DATG. Let $t(S) = \max_{1 \leq k \leq p} t_k(S)$ be the total completion time required to complete the whole computational task according to the schedule S under the precedence constraint. Thus, $t(S)$, from the point of reducing the total computation time, may be used as an objective function for measuring the effectiveness of the schedule S. A smaller $t(S)$ indicates a better schedule S. Thus, a minimum-finishing-time schedule may be defined as the schedule S^* which minimizes $t(S)$, that is,

$$t(S^*) = \min_{S \in \Omega} t(S) = \min_{S \in \Omega} \max_{1 \leq k \leq p} t_k(S) \ . \tag{4}$$

Equation (4) is the optimization criterion for obtaining a minimum finishing time schedule. This means that we want to minimize the maximum processor finishing time, resulting in the so-called minimax optimization criterion. From the definition of D_{cp} (in Eq. (3)), we know that $t(S^*)$ must be greater than or equal to D_{cp}.

Our multiprocessor scheduling problem is to schedule the p processors in a multiprocessor system to complete computing all the modules in the NE task graph (Fig. 2) in a minimum processing time. Any module can be scheduled to be executed on any processor, but each processor can only execute one module at a time. In this paper, we are only interested in the nonpreemptive scheduling, which means that a processor assigned to compute a module is dedicated to that module until it is completed. Furthermore, we assume that the communication time among the processors for data transfer is negligible. Since the solution to the scheduling problem is known to be *NP*-complete, we first solve the problem by an efficient heuristic algorithm, called Dynamical Highest Level First/Most Immediate Successors First (*DHLF/MISF*), for a fast but suboptimal schedule. Next, we use the A^* search algorithm coupled with an heuristic function derived from the Fernandez and Bussell bound to determine an optimal schedule based on the minimax optimization criterion in Eq. (4).

3. Heuristic Scheduling Algorithm

We first propose an efficient heuristic scheduling algorithm *DHLF/MISF* to obtain a fast but suboptimal solution for our multiprocessor scheduling problem for computing the NE equations. Based on the given NE task graph (in Fig. 2), the algorithm constructs a *dynamic* priority list containing all the computational modules arranged in a descending order according to the level of the modules. Similar priority lists were developed by previous methods such as the Highest Level First with Estimated Times (*HLFET*) [16] and the Critical-Path/Most Immediate Successors First (*CP/MISF*) [17] methods. The *HLFET* method constructs a *static* priority list in the descending order of the level of the modules, while the *CP/MISF* method arranges the priority list from the number of the immediate successive modules if the levels of the modules are the same. The suboptimal schedule obtained from these heuristic algorithms starts from zero initially and gradually the modules are "inserted" into the schedule until all the modules have been inserted. Due to the priority lists formed by the *HLFET* and the *CP/MISP* methods are static, they sometime insert unnecessary *null* modules into the processors (i.e. the processors are idle) when a module with a higher level is on the top of the list and thus, the execution of that module must be delayed in order to maintain the precedence constraints. Our proposed dynamic priority list will avoid inserting these unnecessary null modules into the schedule.

Let us denote $A(n)$ be a set of modules that have been assigned to the processors at the nth stage (i.e. the modules that have been inserted into the schedule from the dynamic priority list), and let $\overline{A}(n)$ be the compliment of $A(n)$. Let $P_{mft}(n)$ be the processor(s) with the minimum finishing time at this stage, and $K(n)$ denote the set of modules assigned to the remaining processors but have not finished processing yet. The set $K(n)$ can be explained by the Gantt chart (see Fig. 5). If we conceptually place a vertical "cut-line" at the minimum finishing time, then the modules "cut" by this cut-line are the modules of the set $K(n)$. Let $FW(\overline{A}(n))$ be the function that returns the set of modules, $W(n)$, which are ready to be assigned to all the p processors, i.e., for all $T_i \in \overline{A}(n)$, if and only if $PRED(T_i) \notin \overline{A}(n)$. Similarly, the function $FW(K(n) \bigcup \overline{A}(n)) - K(n)$ returns the set of modules, $R(n)$, which are ready to be assigned to the $P_{mft}(n)$. From these notations, the proposed heuristic algorithm *DHLF/MISF* is described below.

Algorithm DHLF/MISF (*Dynamical Highest Level First/Most Immediate Successors First Algorithm*). Given a task graph, this algorithm constructs a dynamic priority list of all the computational modules and inserts the modules one by one into the suboptimal schedule.

D1. [Initialization.] Initially the schedule is empty (i.e. $A(n) = \varnothing$).

D2. [Determine the levels of modules in $R(n)$] Determine the set of ready modules $R(n)$ and find the level l_i for each module in the set $R(n)$.

D3. [Construct the priority list.] Construct the dynamic priority list in a descending order of l_i. If the levels of the modules are tied, then the module having the largest number of immediately successive modules is assigned to a higher priority.

D4. [Assign the modules.] Assign the modules to the $P_{mfi}(n)$ on the basis of the priority list. If $\bar{A}(n) = \emptyset$, then stop; otherwise, go to **D2**.

END DHLF/MISF.

To demonstrate the efficiency of the proposed heuristic algorithm, a total of 200 random task graphs, each with the number of modules ranging from 10 to 200, was generated for scheduling on a multiprocessor system. Comparison was made between the *DHLF/MISF* solution and the optimal solution by varying the number of processors in the multiprocessor system. Since the optimal solution (i.e. critical-path length) for all the cases can be achieved by increasing the number of processors, the comparison was made between the optimal solution and the solution obtained before the *DHLF/MISF* algorithm reaches the optimal solution. Approximate solutions with a relative error ϵ of less than 5 percent were obtained for 81.5 percent of the cases, and less than 10 percent error for 98.5 percent of the cases. The relative error ϵ is defined as

$$\epsilon = \frac{\text{finishing time of a schdule} - \text{finishing time of the optimal schdule}}{\text{finishing time of the optimal schdule}} . \quad (5)$$

From this computer simulation, all the schedules obtained by this heuristic algorithm for all cases approach to the near-optimal solution. Thus, it is reasonable to use the finishing time obtained by the algorithm *DHLF/MISF* as the upper bound cost of the A^* search algorithm for obtaining the optimal solution. Using the A^* search algorithm for obtaining the optimal solution will be discussed in the next section.

4. State-Space Formulation and A^* Search Algorithm

The optimal scheduling of p processors to compute the robot inverse dynamics to minimize the maximum processor finishing time (Eq. (4)) can be formulated as a state-space search problem. The state-space search paradigm is defined by a triple (U, O, Z), where U is a set of initial states, O is a set of operators on states, and Z is a set of goal states [19,20]. The state space is represented by a search tree in which each node is a state and the application of an operator to a node results in transforming that state to a successor state, a process commonly called node expansion. A solution to the search problem is a path in the state space defined by a sequence of operators which leads a start state to a goal state [18]. In our case of optimal scheduling problem, a solution is an optimal schedule of assigning all the m computational modules to the p processors while minimizing the maximum processor finishing time (in Eq. (4)).

Before we introduce the formulation of the scheduling problem in the state-space representation, we need to define ordered p-tuples and the *MERGE* operation on the ordered p-tuples. Let Q be a set of ordered pairs whose elements indicate the distinct

modules of the desired computational task with their corresponding module execution time (i.e. $Q = \{(T_i, D_i), 1 \leq i \leq m\}$, where T_i is a module and D_i is the corresponding module execution time). We choose j members combinatorially, $j \leq p$, in the set Q to form a set of ordered p-tuples written as $C_j^N(Q)$, where N is the size of the set Q. The elements in each ordered p-tuple are distinctly chosen from the members in the set Q such that the jth member is located at the kth position of the ordered p-tuple, where k, $1 \leq k \leq p$, is the processor number with minimum finishing time at a certain stage of the search tree. Let p_{av} be the number of processors with minimum finishing time. If $p_{av} < p$, then $(p - p_{av})$ elements in the ordered p-tuples are modified to $(\varnothing, x)$, where $\varnothing$ denotes a null-module, and x means that the module execution time is unknown, i.e.

$$C_j^N(Q) = \{< (T_1, D_1), \cdots, (T_i, D_i), (T_j, D_j), \cdots, (T_p, D_p) > \mid (T_i, D_i), (T_j, D_j) \in Q, (T_i, D_i) \neq (T_j, D_j) \text{ except } (T_i, D_i) = (T_j, D_j) = (\varnothing, x)\} .$$

For example, given a set $Q = \{(5,4), (6,5), (7,2), (8,2)\}$ with $p = 3$ processors and both processors 1 and 3 have the minimum finishing time, then the set $C_2^4(Q)$ has six 3-tuples and its elements are $\{<(5,4), (\varnothing, x), (6,5)>, <(5,4), (\varnothing, x), (7,2)>, <(5,4), (\varnothing, x), (8,2)>, <(6,5), (\varnothing, x), (7,2)>, <(6,5), (\varnothing, x), (8,2)>, <(7,2), (\varnothing, x), (8,2)>\}$.

Next we need to define an operation on the ordered p-tuples. Let $T = <(T_1, D_1), (T_2, D_2), \cdots, (T_p, D_p)>$ be an ordered p-tuple as defined previously and $H = <(H_1, J_1)/F_1, (H_2, J_2)/F_2, \cdots, (H_p, J_p)/F_p>$ be an ordered p-tuple with finishing time and F_i is the finishing time of the ith processor. Then, the *MERGE* operation of these two ordered p-tuples, written as $MERGE(T, H)$, results in another ordered p-tuple with finishing time. This means that the module H_i is executed by the processor i followed by the module T_i, and their finishing time is updated and modified to $F_i + D_i$, $1 \leq i \leq p$. Thus, the *MERGE* operation results in merging T and H to form a new p-tuple with finishing time as in

$$MERGE(T, H) = <(T_1, D_1)/(F_1 + D_1), (T_2, D_2)/(F_2 + D_2), \cdots, (T_p, D_p)/(F_p + D_p)> .$$

With this definition and operation on the ordered p-tuples, we are now ready to formulate the state-space search method [19,21] for the minimum-finishing-time scheduling problem as follow.

(1) *State Representation:* States are data structures giving "snapshots" of the condition of the search problem at each stage of its solution. Let an ordered p-tuple with finishing time
$U(n) = <(T_1, D_1)/F_1, \cdots, (T_j, D_j)/F_j, \cdots, (T_p, D_p)/F_p>$ denote a partial schedule at node n in the search tree, which indicates that the module T_j with module execution time D_j at the jth position of the ordered p-tuple is assigned to the jth processor which has a finishing time of F_j.

(2) *Initial State:* The initial state is an empty ordered p-tuple (i.e. no computational modules are assigned to any processors).

(3) *Goal state:* Any state $U(n)$ with $\overline{A}(n) = \varnothing$ is a goal state.

(4) *Operators:* Operators are means for transforming the search problem from one state to another. The application of an operator to a node is to "merge" a new valid ordered p-tuple to $U(n)$. The new valid ordered p-tuples are obtained from a combinatorial selection operation on the set of ready modules $R(n)$ (i.e. $C_j^{N_R}(R(n))$, $1 \leq j \leq p_{av}$, where N_R is the size of the set $R(n)$). While satisfying the precedence constraint of the task graph, the *MERGE* operation updates the ordered p-tuple $U(n)$ by merging a valid ordered p-tuple in the set $C_j^{N_R}(R(n))$, $1 \leq j \leq p_{av}$, to $U(n)$.

The optimal schedule is constructed from the initial state (an empty ordered p-tuple) and gradually the modules of the task are "inserted" into the schedule until they have all been processed. The insertion of ready modules into the schedule is performed by the *MERGE* operation. The *MERGE* operation merges a new valid ordered p-tuple in the set $C_j^{N_R}(R(n))$, $1 \leq j \leq p_{av}$, to $U(n)$. Since our scheduling problem assumes that the module execution time D_i's are different from one another, there exists the possibility that the optimal schedule can not be obtained simply by assigning all the ready modules to the processors at the same time [15]. To determine the optimal schedule when all the D_i's are different, it is not suffice to generate the number of successive nodes N_{n_s} that equals to the number of combinations from the ready modules. Instead, at each successive node generating procedure, assigning null modules to processors may lead to a better schedule. This null-module assignment together with the ready-module assignment must be considered in our scheduling problem to determine an optimal schedule. Thus, the number of successive nodes N_{n_s} generated at each node n is given by

$$N_{n_s} = \begin{cases} \sum_{i=1}^{p_{av}} \binom{N_R}{i} + 1, & \text{if } p_{av} < p \\ \sum_{i=1}^{p_{av}} \binom{N_R}{i}, & \text{if } p_{av} = p \end{cases}, \tag{6}$$

and the modules in each combination form an ordered p-tuple. Hence, $C_j^{N_R}(R(n))$, $1 \leq j \leq p_{av}$, is the set which contains all the possible ordered p-tuples for all the combinations. Note that if $N_R < p_{av}$, then N_{n_s} reduces to 1. A next state generation algorithm (*NSG*) is developed to generate the successive nodes and the next state in the search tree.

Algorithm NSG (*Next State Generation*). This algorithm generates the successive nodes and the next state in the search tree.

N1. [Initialization and determine $R(n)$.] Find the set of ready modules $R(n)$ and set $N_R = |R(n)|$.

N2. [Level ordering and indexing.] Based on the descending order of the level l_i and the number of immediate successive modules in the set $R(n)$, index the module number in an ascending order. Set $j = p_{av}$.

N3. [Select j elements from $R(n)$ combinatorially.] According to the index number, choose j elements in the set $R(n)$ lexicographically to obtain the set $C_j^{N_R}(R(n))$.

N4. [Looping.] If $j \geq 1$, then set $j = j-1$ and go to **N3**; otherwise continue.

N5. [Obtain the new states.] Apply the *MERGE* operation to a valid p-tuple in the set $C_j^{N_R}(R(n))$ generated from **N3** and **N4** to the elements in the ordered p-tuple $U(n)$, resulting in new states.

END NSG

The above formulation presents a state-space search formalism in which a scheduling solution can be obtained. The path from the start node to a goal node corresponds to a scheduling solution. The cost defined on each node expansion is according to Eq. (15). The minimum-finishing-time-scheduling solution is the path from a start node to a goal node with the minimum cost path.

In general, in a state-space search, the number of nodes expanded before reaching a solution is likely to be prohibitively large, usually combinatorially explosive. Furthermore, in our case, the search tree could be enormous if the number of processors and the number of modules are large. Certainly, the so-called "blind search" (i.e. the order of potential solution paths considered is arbitrary, using no information to judge where the solution is likely to be) should not be used in searching for a solution for our scheduling problem. But rather an ordered search which utilizes heuristic information about the search problem to reduce the number of nodes expanded should be used to expand the "most promising" node to achieve the optimal path from the start node to the goal node.

Among all the ordered search algorithms, the well-known A^* algorithm [18] will be used to find a scheduling solution. The A^* algorithm is guaranteed to be optimal, if the evaluation function $f(n)$ for node expansion is properly defined. The use of an evaluation function is to speed up the search process by properly ordering most promising nodes for expansion, that is, the node selected for expansion is the one with minimum $f(n)$. An evaluation function $f(n)$ at any node n estimates the sum of the cost of the minimal cost path from the start node to node n plus the cost of a minimal cost path from node n to a goal node [18]. Thus, $f(n)$ is an estimate of the cost of a minimal cost path constrained to go through node n and can be defined as

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

where $g(n)$ is the cost of the minimal cost path from the start node to node n in the state space and $h(n)$ is an estimate of the cost of the minimal cost path $h^*(n)$ from node n to a goal node. The A^* algorithm can be considered to be a branch-and-bound search using the dynamic programming principle with a cost estimate of remaining unassigned modules [20]. For our scheduling problem, using the above formulation and definitions, we can obtain $g(n)$ by consecutively applying the *MERGE* operation from an initial node to node n.

As to the heuristic function $h(n)$, physical meaning about the scheduling problem can be used to define and select an appropriate $h(n)$. The objective is to design and construct $h(n)$ to be a close estimate to the true $h^*(n)$ for all n without overestimating $h^*(n)$. If the heuristic function $h(n)$ is overestimating $h^*(n)$, the ordered search may miss an optimal solution or all solutions. On the other extreme, if $h(n) = 0$ for all n, no heuristic information about the search problem is used to order the node expansion and the search reduces to a uniform-cost search. Thus, in order to speed up the search process and reduce the number of nodes expanded, nonzero lower bound estimate of $h^*(n)$ should be chosen.

In our A^* algorithm, a heuristic function based on the Fernandez and Bussell (*F&B*) bound will be used [22]. The Fernandez and Bussell bound indicates the lower bound of the minimum-finishing-time schedule for a fixed number of processors and is given by the function

$$t_L = D_{cp} + \lceil q_e \rceil \,, \tag{8}$$

where

$$q_e = \max_{0 \le t_k \le D_{cp}} [-t_k + \frac{1}{p} \int_0^{t_k} \psi(\bar{\tau}, t)\, dt \,], \tag{9}$$

and $\psi(\bar{\tau}, t)$ is the load density function and is given by

$$\bar{\tau}_j = D_{cp} - l_j \tag{10a}$$

$$\pi(\bar{\tau}_j, t) = \begin{cases} 1 \,, & \text{for } t \in [\bar{\tau}_j, \bar{\tau}_j + D_j] \\ 0 \,, & \text{otherwise} \end{cases} \tag{10b}$$

$$\psi(\bar{\tau}, t) = \sum_{j=1}^{m} \pi(\bar{\tau}_j, t) \,. \tag{10c}$$

Since the *F&B* lower bound is only valid for the whole DATG, it needs to be modified in order to be used for a sub-DATG in our scheduling problem. Consider the Gantt chart of a schedule at node n of the search tree with p processors shown in Fig. 5. Let $A_k(n)$ be the set of assigned modules executed by processor(s) with the minimum

finishing time, $F_{\min}(n)$. Let $K(n)$ be a set of modules T_j, $T_j \notin A_k(n)$, assigned to be executed by the remaining processors, in which the jth processor has a finishing time of $F_j(n)$ and $F_j(n) > F_{\min}(n)$, $1 \le j \le p$. Since the cut-line of the Gantt chart is located at the $F_{\min}(n)$, for any module whose finishing time $F_j(n)$ is greater than $F_{\min}(n)$, the remaining execution time is $(F_j(n) - F_{\min}(n))$ time unit, and $(D_j - (F_j(n) - F_{\min}(n)))$ execution time has elapsed for the module T_j, $1 \le j \le p$. This results in changing the level of the modules in the set $K(n)$. We denote the pseudo-level of the module T_j, $\hat{l}_j(n)$, as the level of the original level subtracts the elapsed execution time units. In other words, $\hat{l}_j(n) = l_j(n) - (D_j - (F_j(n) - F_{\min}(n)))$, $1 \le j \le p$, $T_j \in K(n)$. Then $K(n) \bigcup \bar{A}(n)$ is the set of modules needed to be scheduled if the cut-line is at the $F_{\min}(n)$. Note that this set of modules includes ready modules, null modules, and unfinished modules at the $F_{\min}(n)$. The execution time of T_j, $T_j \in K(n)$, is updated and modified to $F_j(n) - F_{\min}(n)$. This in effect is equivalent to splitting the assigned module T_j, $T_j \in K(n)$, into two parts: the first part is executed by the jth processor, the second part with execution time $(F_j(n) - F_{\min}(n))$ is combined with $\bar{A}(n)$ to calculate the *F&B* bound as an estimate of $h^*(n)$. Note that this splitting of the assigned modules is only used to calculate the *F&B* bound and no physical splitting is taken place. Thus, the heuristic function can be written as

$$h(n) = \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j) + \lceil q_{ec} \rceil, \tag{11}$$

where

$$q_{ec} = \max_{0 \le t_k \le \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j)} [-t_k + \frac{1}{p} \int_0^{t_k} \psi_c(\bar{\tau}, t) dt], \tag{12}$$

and $\psi_c(\bar{\tau}, t)$ is given by

$$\bar{\tau}_j = \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j) - y_j \tag{13a}$$

$$y_j = \begin{cases} \hat{l}_j, & \text{if } T_j \in K(n) \\ l_j, & \text{otherwise} \end{cases} \tag{13b}$$

$$\pi_c(\bar{\tau}_j, t) = \begin{cases} 1, & \text{for } t \in [\bar{\tau}_j, \bar{\tau}_j + D_j'] \\ 0, & \text{otherwise} \end{cases} \tag{13c}$$

$$D_j' = \begin{cases} F_j(n) - F_{\min}(n), & \text{for } T_j \in K(n) \\ D_j, & \text{otherwise} \end{cases} \tag{13d}$$

$$\psi_c(\bar{\tau}, t) = \sum_{T_j \in K(n) \bigcup \bar{A}(n)} \pi_c(\bar{\tau}_j, t). \tag{13d}$$

Note that the level of the modules in the set $K(n)$ is updated to $\hat{l}_j(n)$. Our heuristic function in Eq. (11) is sharper than the Kasahara and Narita's heuristic function. A comparison of our heuristic function with the Kasahara and Narita's heuristic function is discussed in the Appendix A. We can further improve the performance of our heuristic function in Eq. (11) by modifying the pseudo-level of the modules in the set $K(n)$ to $\max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j)$. This is stated in the following proposition.

Proposition 1: If $\hat{l}_j(n) = \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j)$, $T_j \in K(n)$, in Eq. (13b), then the heuristic function is a better estimate than equation (11).

Proof : Since the schedule is non-preemptive, the modules in the set $K(n)$ must be executed immediately. From the definition of the latest completion time of the modules and the latest vertex activity function [22] in Fig. 6c, the physical meaning of changing $\hat{l}_j(n)$ to $\max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j)$ is to move the shaded areas from the interval $[\bar{\tau}_j, \bar{\tau}_j + D_j']$ to the interval $[0, D_j']$. After this movement, the area function $AREA_c(0, t_x)$† remains unchanged, where $t_x = \max_{T_j \in K(n)} (\bar{\tau}_j + D_j')$. However, the area function $AREA_c(0, t_y)$, $0 < t_y < t_x$, has been increased by the moved area. Thus, the heuristic function with $\hat{l}_j(n) = \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j)$ in Eq. (11) has a better estimate.

□

Since the *F&B* bound underestimates the finishing time of the DATG and the minimum time taken to process the DATG by using the preemptive schedule is the lower bound of the non-preemptive schedule [22], the heuristic function in Eq. (11) is admissible. Hence, for our scheduling problem, using previous formulation and Eq. (4), we have

$$g(n) = \max_{1 \le k \le p} t_k(S) - \max_{T_j \in K(n)} D_j' = F_{\min}(n) \tag{14}$$

where D_j' is as in Eq. (13d). Then the evaluation function for the A^* algorithm becomes

$$f(n) = g(n) + h(n) \tag{15}$$

where $h(n)$ is stated as in Eq. (11) and $\hat{l}_j(n) = \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j)$.

---

† $AREA_c(0, t_x) \triangleq \int_0^{t_x} \psi_c(\bar{\tau}, t) dt$.

With the above heuristic function, the MIN_LENGTH algorithm has been developed to find the minimum-finishing-time schedule within a specified relative error ϵ (Eq. (5)). In this algorithm, the finishing time obtained by the heuristic algorithm *DHLF/MISF* will be used as the upper bound cost of the evaluation function of the A^* search algorithm for obtaining the optimal solution. In the node expansion process, whenever the evaluation function of a node in the search tree is greater than this upper bound, then this node is pruned from the *OPEN* list. This is because the node contributes no better solution than the *DHLF/MISF* heuristic method. This pruning greatly reduces the time and space complexity of the minimum-cost search in the *OPEN* list.

Algorithm MIN_LENGTH (*Minimum-Length Schedule Algorithm*). This algorithm determines the minimum-finishing-time schedule within the specified relative error ϵ.

Input: A DATG of the desired computational task, the upper bound of finishing time, *UB*, obtained from the *DHLF/MISF* heuristic algorithm, and the desired relative error ϵ of the solution schedule.

Output: An optimal schedule with minimum finishing time.

M1. [Initialization.] Create an empty *OPEN*1 list. Put the initial node I on a list of unexpanded nodes called *OPEN*. Calculate the evaluation function $f(I)$ in Eq. (15). Note that if the initial state I is an empty set, then the value $f(I)$ is equal to the *F&B* bound. If $UB = f(I)$, then exit and an optimal solution has been found by the *DHLF/MISF* heuristic algorithm; otherwise continue.

M2. [Find the "*best node*" from *OPEN* list with minimum cost.] Select from the *OPEN* list a node n with minimum $f(n)$. If several nodes qualify, choose a goal node if there is one, otherwise choose the node on the top of the *OPEN* list.

M3. [Move node from unexpanded list to expanded list.] Remove node n from the *OPEN* list and place it on a list of expanded nodes called *CLOSED*.

M4. [Check goal node.] If n is a goal node (i.e. $\bar{A}(n) = \varnothing$), exit with success and the optimal solution has been determined; otherwise continue. (If $\epsilon \neq 0$, then a suboptimal solution has been determined.)

M5. [Expand node n.] Use the *MERGE* operation on node n and create all its successor nodes by using the algorithm *NSG*. Generate all its successor nodes n_s and place them on the top of the *OPEN*1 list.

M6. [Check evaluation function.] If *OPEN*1 $\neq$ an empty list, calculate $f(n_s)$ of the node on the top of the *OPEN*1 list according to Eq. (15).

a) If $\lceil f(n_s)/(1+\epsilon) \rceil \leq f(n)$, remove n_s from the *OPEN*1 list and place it on the *CLOSED* list. Set $n = n_s$ and go to **M4.**

b) If $UB > \lceil f(n_s)/(1+\epsilon) \rceil > f(n)$, place this node on the bottom of the *OPEN* list.

c) If $UB \leq \lceil f(n_s) / (1+\epsilon) \rceil$, prune this node.

M7. [Loop.] If $OPEN1$ = empty list, then go to **M2**; otherwise go to **M6**.

END MIN_LENGTH

Note that besides the *OPEN* and *CLOSED* lists in the traditional A^* algorithm [18], we use another $OPEN1$ list which contains the nodes without calculating the evaluation function. The next node expansion is chosen from the node with minimum cost in the *OPEN* list. This greatly reduces the time and computational complexity in calculating all the evaluation function of all the successive nodes and the minimum-cost search in the *OPEN* list.

As an example, consider the minimum-finishing-time schedule of a given DATG as shown in Fig. 4a ($m = 9$). The computational modules are to be executed by 2 identical processors ($p = 2$). The level number of each module is given beside each module in the task graph. We used the above MIN_LENGTH algorithm to determine an optimal schedule. The optimal schedule is the path from the initial node to the goal node with minimum cost path as shown in Fig. 4b and the minimum-scheduled length is found to be 16 time units. In determining the optimal schedule, node expansion is based on the minimum value of the evaluation function $f(n)$ in Eq. (15). The $U(n)$, $\lceil q_{ec} \rceil$, $f(n)$, and $R(n)$ associated with the node expansion are listed in Table 4. For comparison, we also determined a suboptimal schedule using the proposed heuristic algorithm *DHLF/MISF*. The suboptimal schedule length is found to be 17 time units. The Gantt charts for both schedules are shown in Fig. 7.

As shown in Fig. 4b, a total of 8 nodes are expanded in the search tree and 22 nodes are generated before the goal node is found. If we use the critical path as the heuristic function $h(n) = D_{cp}(n)$ for all nodes to find an optimal schedule, a total of 10 nodes are expanded and 30 nodes are generated before the optimal schedule is found. Since the nodes generated in the scheduling problem is combinatorial explosive, the use of the heuristic function $h(n)$ greatly reduces the number of node expansions. For this specific example, the number of node expansions and the number of nodes generated have been reduced approximately by 25% and 36%, respectively.

5. Computer Simulation

The proposed A^* algorithm and the heuristic algorithm *DHLF/MISF* were used to schedule the inverse dynamics computation of a Stanford arm on a multiprocessor system [8,9]. The recursive Newton-Euler equations of motion were used to compute the inverse dynamics and a task graph for this computational task can be found in [8,9]. Using this task graph, Luh and Lin, and Kasahara and Narita were able to shorten the required computation time to 9.67 *ms* and 5.73 *ms*, respectively, with six processors. Based on the same task graph, our proposed A^* algorithm shows further improvement in the computational time and the number of processors used to achieve the critical-path-length computation. Using $p = 6$ processors, our A^* algorithm achieves the

critical-path-length computation of 5.70 ms, which means that the use of more than 6 processors for parallel processing will never obtain a shorter processing time. Kasahara and Narita's scheduling algorithm requires $p = 7$ processors to achieve the same critical-path-length computation of 5.70 ms, while Luh and Lin, although using 6 processors, were not able to achieve this minimum-time computation. The optimal allocation of modules in each of the 6 processors is listed in Table 5. Table 6 details the computer simulation results of our A^* scheduling algorithm as compared to Luh and Lin's [8] and Kasahara and Narita's results [9].

To further validate the efficiency of our proposed A^* algorithm and the heuristic algorithm *DHLF/MISF*, we would like to apply them to compute the inverse dynamics of any n-link manipulator with rotary/prismatic joints. Since the recursive Newton-Euler equations of motion are applicable to manipulators with rotary/prismatic joints, they can be expressed in the HLR form (with respect to the base coordinate system). This results in achieving an efficient task graph for computing the inverse dynamics of any n-link manipulator with rotary/prismatic joints. This task graph is shown in Fig. 2. For a 6-link, PUMA-like manipulator, this task graph shows that the NE equations can be decomposed into 606 computational modules. Based on the task graph in Fig. 2, our A^* algorithm and the heuristic algorithm *DHLF/MISF* are used to schedule the computation of the modules in the task graph on a multiprocessor system whose primitive processing elements are constructed by a group of modular processors (MPs). Each of these MPs has a microprocessor-like architecture. Each MP can evaluate the operation of 3×1 vector addition or vector dot product simultaneously. In this computer simulation, we used three Motorola MC68020 microprocessors running at a clock rate of 16.7 MHz to form a modular processor. The MC68020 microprocessor takes 3 clock cycles ($0.2\,\mu s$) and 30 clock cycles ($2\,\mu s$), respectively, to compute one floating-point addition and multiplication. The optimal schedules for any number of MPs determined by our A^* algorithm are listed in Table 7. From Table 7, our A^* algorithm indicates that, using 38 MPs (or 114 microprocessors), the critical-path-length computation can be achieved for a 6-link, PUMA-link manipulator. This translates to 31 additions and 9 multiplications which lead to a processing time of $24.2\,\mu s$. If six microprocessors are used, then the optimal schedule requires a computation time of $400.4\,\mu s$. Note that the above computation time does not include data acquisition, data scaling, and the interprocessor communication time. We also used the heuristic *DHLF/MISF* algorithm to obtain fast but suboptimal schedules. In Table 7, the relative error ϵ indicates the power and efficiency of our heuristic algorithm.

6. Conclusion

The A^* algorithm and the heuristic algorithm *DHLF/MISF* were proposed to determine the minimum-length scheduling problem on a multiprocessor system for computing the inverse dynamics of an n-link manipulator with rotary/prismatic joints. Minimizing the maximum processor finishing time is used as an objective function for

the scheduling optimization. Although maximum parallelism task graphs can be obtained for the NE equations of motion expressed either in the HLR or the IHLR form, the NE task graph in the HLR form results in minimum arithmetic operations. For a 6-link, PUMA-like manipulator, this task graph shows that the NE equations can be decomposed into 606 computational modules. The problem of determining an optimal schedule consisting of m modules and p processors is usually combinatorial explosive. Our use of the heuristic function $h(n)$, based on the Fernandez and Bussell bound, in the evaluation function $f(n)$ of the A^* algorithm greatly reduces the time complexity. Computer simulation results indicate that the proposed A^* algorithm and the heuristic algorithm *DHLF/MISF* are efficient and practical that they can provide suboptimal as well as optimal solutions. Our A^* algorithm indicates that, using 38 MPs or 114 microprocessors, the critical-path-length computation can be achieved for a 6-link, PUMA-link manipulator. This translates to 31 additions and 9 multiplications which lead to a processing time of 24.2 μs, if MC68020 microprocessors running at 16.7 *MHz* clock rate are used. If six MC68020 microprocessors are used, then the optimal schedule requires a computation time of 400.4 μs.

Appendix A

This appendix proves that our heuristic function in Eq. (11) is shaper than the Kasahara and Narita's heuristic function [9]. The heuristic function of Kasahara and Narita ($K\&N$) can be written as

$$h_{K\&N}(n) = \max_{T_j \in \bar{A}(n)} l_j + \lceil q_{eK\&N} \rceil \tag{16}$$

where

$$q_{eK\&N} = \max_{0 \le t_k \le \max_{T_j \in \bar{A}(n)} l_j} [-t_k + \frac{1}{p} \int_0^{t_k} \psi(\bar{\tau}, t) dt \,] \tag{17}$$

and $\psi(\bar{\tau}, t)$ is given by

$$\bar{\tau}_j = \max_{T_j \in \bar{A}(n)} l_j - l_j \tag{18a}$$

$$\pi(\bar{\tau}_j, t) = \begin{cases} 1, & \text{for } t \in [\bar{\tau}_j, \bar{\tau}_j + D_j \,] \\ 0, & \text{otherwise} \end{cases} \tag{18b}$$

$$\psi(\bar{\tau}, t) = \sum_{T_j \in \bar{A}(n)} \pi(\bar{\tau}_j, t \,). \tag{18c}$$

The $K\&N$ heuristic function is computed from the modules in the set $\bar{A}(n)$, while our heuristic function is calculated from the modules in the set $K(n) \bigcup \bar{A}(n)$. Two cases can be identified. First, if $\max_{T_j \in K(n)} \hat{l}_j \le \max_{T_j \in \bar{A}(n)} l_j$, then $\max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j) = \max_{T_j \in \bar{A}(n)} l_j$. From Eq. (11), we know that the intergration includes $(F_j(n) - F_{\min}(n))$ time units in the function $\pi_c(\bar{\tau}_j, t)$. Thus, $\lceil q_{ec} \rceil \ge \lceil q_{eK\&N} \rceil$ and $h(n) \ge h_{K\&N}(n)$. Second, if $\max_{T_j \in K(n)} \hat{l}_j > \max_{T_j \in \bar{A}(n)} l_j$, then $\max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j) > \max_{T_j \in \bar{A}(n)} l_j$. Since $\lceil q_e \rceil$ is a nonlinear function, it is difficult to judge whether $\lceil q_{ec} \rceil$ or $\lceil q_{eK\&N} \rceil$ is larger. Thus, we need to determine that $\max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j) + \lceil q_{ec} \rceil \ge \max_{T_j \in \bar{A}(n)} l_j + \lceil q_{eK\&N} \rceil$. Consider the latest vertex activity function $\pi(\bar{\tau}, t)$ of a given interval $[0, D_{cp} \,]$ as described in Fig. 6a. At a certain stage of the search tree, Eqs.(16) and (11) can be represented by Figs. 6b and 6c, respectively. The shaded areas in Figs. 6b and 6c represent the modules $T_j \notin \bar{A}(n)$ and $T_j \notin K(n) \bigcup \bar{A}(n)$, respectively. Eq. (16) can be further written as

$$h_{K\&N}(n) = \begin{cases} \max\limits_{T_j \in \bar{A}(n)} l_j , & \text{if } \frac{1}{p} AREA_{K\&N}(0, t_k) - t_k > \max\limits_{T_j \in \bar{A}(n)} l_j , \\ & 0 \le t_k \le \max\limits_{T_j \in \bar{A}(n)} l_j \\ \lceil \max\limits_{T_j \in \bar{A}(n)} l_j - t_k + \frac{1}{p} AREA_{K\&N}(0, t_k) \rceil, & \text{otherwise .} \end{cases} \tag{19}$$

Consider the activity function in Fig. 6, the value $t_k = 0$ in Fig. 6b corresponds to the value $t_k = \max\limits_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j , \hat{l}_j) - \max\limits_{T_j \in \bar{A}(n)} l_j$ in Fig. 6c. Thus, $\max\limits_{T_j \in \bar{A}(n)} l_j - t_k$, $0 \le t_k \le \max\limits_{T_j \in \bar{A}(n)} l_j$, is equal to $\max\limits_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j , \hat{l}_j) - t_k$, $\max\limits_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j , \hat{l}_j) - \max\limits_{T_j \in \bar{A}(n)} l_j \le t_k \le \max\limits_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j , \hat{l}_j)$. Since the area function $AREA_c(0, t_k)$, $0 \le t_k \le \max\limits_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j , \hat{l}_j)$, in Fig. 6c is greater than the area function $AREA_{K\&N}(0, t_k)$, $0 \le t_k \le \max\limits_{T_j \in \bar{A}(n)} l_j$, we have $\max\limits_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j , \hat{l}_j) + \lceil q_{ec} \rceil \ge \max\limits_{T_j \in \bar{A}(n)} l_j + \lceil q_{eK\&N} \rceil$.

Appendix B

The procedure of evaluating the Newton-Euler equations of motion expressed as a HLR form is given below:

1) Compute module T_1, the 3×3 rotation matrices of link i coordinates with respect to the base coordinates, ${}^0\mathbf{R}_i$, $i = 1,2, \cdots ,n$

$$ {}^0\mathbf{R}_i = {}^0\mathbf{R}_{i-1} \, {}^{i-1}\mathbf{R}_i $$

2) Compute modules T_2 and T_3,

$$ b_i = \mathbf{z}_{i-1} \dot{q}_i (1 - \lambda_i) $$

and

$$ \omega_i = \omega_{i-1} + b_i $$

3) Compute module T_4,

$$ \mathbf{z}_i = {}^0\mathbf{R}_i \, \mathbf{z}_0 \quad , \quad \mathbf{z}_0 = [0,0,1]^T $$

$$ \mathbf{p}_i^* = {}^0\mathbf{R}_i \, {}^i\mathbf{p}_i^* $$

$$ \mathbf{s}_i = {}^0\mathbf{R}_i \, {}^i\mathbf{s}_i $$

The evaluation of $\mathbf{z}_i$ only involves taking the third column of ${}^0\mathbf{R}_i$.

4) Compute modules T_5 and T_6,

$$b_i = (\mathbf{z}_{i-1}\,\ddot{q}_i + \omega_{i-1} \times \mathbf{z}_{i-1}\,\dot{q}_i)\,(1-\lambda_i)$$

and

$$\dot{\omega}_i = \dot{\omega}_{i-1} + b_i$$

5) Compute modules T_7 to T_{11},

$$b_i = \dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) + (\mathbf{z}_{i-1}\,\ddot{q}_i + 2\,\omega_i \times (\mathbf{z}_{i-1}\,\dot{q}_i))\,\lambda_i$$

and

$$\ddot{\mathbf{p}}_i = \ddot{\mathbf{p}}_{i-1} + b_i$$

6) Compute modules T_{12} to T_{15},

$$\ddot{\mathbf{r}}_i = \dot{\omega}_i \times \mathbf{s}_i + \omega_i \times (\omega_i \times \mathbf{s}_i) + \ddot{\mathbf{p}}_i$$

7) Compute module T_{16},

$$\mathbf{F}_i = m_i \ddot{\mathbf{r}}_i$$

8) Compute module T_{17},

$$\mathbf{f}_i = \mathbf{f}_{i+1} + \mathbf{F}_i$$

9) Compute modules T_{20} to T_{26},

$$\mathbf{N}_i = \mathbf{J}_i\,\dot{\omega}_i + \omega_i \times (\mathbf{J}_i\,\omega_i)$$

For the sake of saving the calculations of evaluating $\mathbf{J}_i = {}^0\mathbf{R}_i\ {}^i\mathbf{J}_i\ {}^i\mathbf{R}_0$, the above equation is modified to

$${}^i\omega_i = {}^i\mathbf{R}_0\,\omega_i = ({}^0\mathbf{R}_i)^T\,\omega_i$$

$${}^i\dot{\omega}_i = {}^i\mathbf{R}_0\,\dot{\omega}_i = ({}^0\mathbf{R}_i)^T\,\dot{\omega}_i$$

$${}^i\mathbf{N}_i = {}^i\mathbf{J}_i\ {}^i\dot{\omega}_i + {}^i\omega_i \times ({}^i\mathbf{J}_i\ {}^i\omega_i)$$

$$\mathbf{N}_i = {}^0\mathbf{R}_i\ {}^i\mathbf{N}_i$$

10) Compute modules T_{18}, T_{19}, T_{27} to T_{29},

$$b_i = \mathbf{N}_i + (\mathbf{p}_i^* + \mathbf{s}_i) \times \mathbf{F}_i + \mathbf{p}_i^* \times \mathbf{f}_{i+1}$$

and

$$\mathbf{n}_i = \mathbf{n}_{i+1} + b_i$$

11) Compute module T_{30},

$$\tau_i = \begin{cases} (\mathbf{n}_i)^T \mathbf{z}_{i-1} & , \text{ if } \lambda_i = 0 \\ (\mathbf{f}_i)^T \mathbf{z}_{i-1} & , \text{ if } \lambda_i = 1 \end{cases}$$

Previously undefined terms, expressed in the base coordinates, are given as follows:

m_i is the mass of link i,

ω_i is the angular velocity of link i,

$\dot{\omega}_i$ is the angular acceleration of link i,

$\ddot{\mathbf{p}}_i$ is the linear acceleration of link i,

$\ddot{\mathbf{r}}_i$ is the linear acceleration of the center of mass of link i,

$\mathbf{F}_i$ is the total force exerted on link i at the center of mass,

$\mathbf{N}_i$ is the total moment exerted on link i at the center of mass,

$\mathbf{f}_i$ is the force exerted on link i by link $i-1$,

$\mathbf{n}_i$ is the moment exerted on link i by link $i-1$,

τ_i is the torque exerted by the actuator at joint i if rotational, force if translational,

q_i is the joint variable of joint i (θ_i if rotational and d_i if translational).

Appendix C

The procedure of evaluating the Newton-Euler equations of motion expressed as a IHLR form is given below:

1) Compute modules T_1 and T_2,

$$b_i = {}^i\mathbf{R}_{i-1}\, \mathbf{z}_0 \dot{q}_i\, (1 - \lambda_i)$$

and

$$\omega_i = {}^i\mathbf{R}_{i-1}\, \omega_{i-1} + b_i$$

2) Compute modules T_3 to T_6,

$$b_i = (1 - \lambda_i)\, {}^i\mathbf{R}_{i-1} (\mathbf{z}_0\, \ddot{q}_i + \omega_{i-1} \times \mathbf{z}_0\, \dot{q}_i)$$

and

$$\dot{\omega}_i = {}^i\mathbf{R}_{i-1}\,\dot{\omega}_{i-1} + b_i$$

3) Compute modules T_7 to T_{13},

$$b_i = \dot{\omega}_i \times \mathbf{p}_i^* + \omega_i \times (\omega_i \times \mathbf{p}_i^*) + \lambda_i\,({}^i\mathbf{R}_{i-1}\,\mathbf{z}_0\,\ddot{q}_i + 2\,\omega_i \times ({}^i\mathbf{R}_{i-1}\,\mathbf{z}_0\,\dot{q}_i))$$

and

$$\ddot{\mathbf{p}}_i = {}^i\mathbf{R}_{i-1}\,\ddot{\mathbf{p}}_{i-1} + b_i$$

4) Compute modules T_{14} to T_{17},

$$\ddot{\mathbf{r}}_i = \dot{\omega}_i \times \mathbf{s}_i + \omega_i \times (\omega_i \times \mathbf{s}_i) + \ddot{\mathbf{p}}_i$$

5) Compute module T_{18},

$$\mathbf{F}_i = m_i\,\ddot{\mathbf{r}}_i$$

6) Compute module T_{19},

$$\mathbf{f}_i = {}^i\mathbf{R}_{i+1}\,\mathbf{f}_{i+1} + \mathbf{F}_i$$

7) Compute modules T_{20} to T_{23},

$$\mathbf{N}_i = \mathbf{J}_i\,\dot{\omega}_i + \omega_i \times (\mathbf{J}_i\,\omega_i)$$

8) Compute modules T_{24} to T_{29},

$$b_i = \mathbf{N}_i + (\mathbf{p}_i^* + \mathbf{s}_i) \times \mathbf{F}_i + \mathbf{p}_i^* \times \mathbf{f}_{i+1}$$

and

$$\mathbf{n}_i = {}^i\mathbf{R}_{i+1}\,\mathbf{n}_{i+1} + b_i$$

9) Compute module T_{30},

$$\tau_i = \begin{cases} (\mathbf{n}_i)^T\,\mathbf{z}_{i-1} & , \text{ if } \lambda_i = 0 \\ (\mathbf{f}_i)^T\,\mathbf{z}_{i-1} & , \text{ if } \lambda_i = 1 \end{cases}$$

7. References

[1] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "On-line Computational Scheme for Mechanical Manipulator," *Trans. ASME, J. Dynam., Syst., Meas., Contr.,* Vol. 120, pp. 69-76, June 1980.

[2] D. E. Orin, R. B. MaChee, M. Vukobratovic, and G. Hartoch, "Kinematics and Kinetic Analysis of Open-Chain Linkages Utilizing Newton-Euler Methods," *Math. Biosci.,* Vol. 43, pp. 107-130, 1979.

[3] C. S. G. Lee, T. N. Mudge, and J. L. Turney, "Hierarchical Control Structure Using Special Purpose Processor for the Control of Robot Arm," *Proc. 1982 Conf. Patt. Recog. and Image Processing,* pp. 634-640, June 1982.

[4] R. Nigam and C. S. G. Lee, "A Multiprocessor-Based Controller for Control of Mechanical Manipulators," *IEEE J. of Robotics and Autom.,* Vol. RA-1, No. 4, pp. 173-182, Dec. 1985

[5] T. Kanade, P. K. Khosla, and N. Tanaka, "Real-Time Control of the CMU Direct Arm II Using Customized Inverse Dynamics," *Proc. of IEEE Conf. on Decision and Contr.,* pp. 1345-1352, Dec. 1984.

[6] L. H. Lathrop, "Parallelism in Manipulator Dynamics," MIT Artificial Intelligence Lab., Cambridge, MA, Tech. Rep. No. 754, Dec. 1983.

[7] C. S. G. Lee and P. R. Chang, "Efficient Parallel Algorithm for Robot Inverse Dynamics Computation," *IEEE Trans. Syst. Man, and Cybern.,* Vol. SMC-16, No. 4, pp. 532-542, July 1986.

[8] J. Y. S. Luh and C. S. Lin, "Scheduling of Parallel Computer for a Computer-Controlled Mechanical Manipulator," *IEEE Trans. Syst. Man and Cybern.,* Vol. 12, pp. 214-234, 1982.

[9] H. Kasahara and S. Narita, "Parallel Processing of Robot-Arm Control Computation on a Multiprocessor System," *IEEE J. of Robotics and Autom.,* Vol. RA-1, No. 2, pp. 104-113, June 1985.

[10] J. Barhen, "Robot Inverse Dynamics on a Concurrent Computation Ensemble," *Proc. of 1985 ASME Int'l Conf. on Computers in Engineering,* Vol. 3, pp. 415-429, 1985.

[11] E. G. Coffman, *Computer and Job-Shop Scheduling Theory,* Wiley, New York, 1976.

[12] J. K. Lenstra and A. H. G. R. Kan, "Complexity of Scheduling Under Precedence Constraints," *Oper. Res.,* Vol. 26, pp. 25-35, Jan. 1978.

[13] M. J. Gonzalez, Jr., "Deterministic Processor Scheduling," *Computing Surveys,* Vol. 9, No. 3, 1977.

[14] M. R. Garey, R. L. Graham, and D. J. Johnson, "Performance Guarantees for Scheduling Algorithm," *Oper. Res.,* Vol. 26, pp. 3-21, Jan. 1978.

[15] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez. "Optimal Scheduling Strategies in a Multiprocessor System," *IEEE Trans. Computers,* Vol. C-21, pp. 137-146, Feb. 1972.

[16] T. L. Adam, K. M. Chandy and J. R. Dickson, "A Comparison of List schedules for Parallel Processing Systems," *Commun. Ass. Comput. Mach.,* Vol. 17, pp. 685-690, Dec. 1974.

[17] H. Kasahara and S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," *IEEE Trans. Computers,* Vol. C-33, pp. 1023-1029, Nov. 1984.

[18] N. J. Nilsson, *Principle of Artificial Intelligence,* Tioga Publishing Company, Palo Alto, CA, 1980.

[19] A. Barr and E. A. Feigenbaum, *The Handbook of Artificial Intelligence,* Vol. 1, William Kaufman, Los Altos, CA, 1981.

[20] P. H. Winston, *Artificial Intelligence,* Addison-Wesley, 1984

[21] C. C. Shen and W. H. Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing System Using a Minimax Criterion," *IEEE Trans. Computers,* Vol. C-34, pp. 745-751, Mar. 1985.

[22] E. B. Fernandez and B. Bussell, "Bound on the Number of Processors and Time for Multiprocessor Optimal Schedules," *IEEE Trans. Computers,* Vol. C-22, pp. 745-751, Aug. 1973.

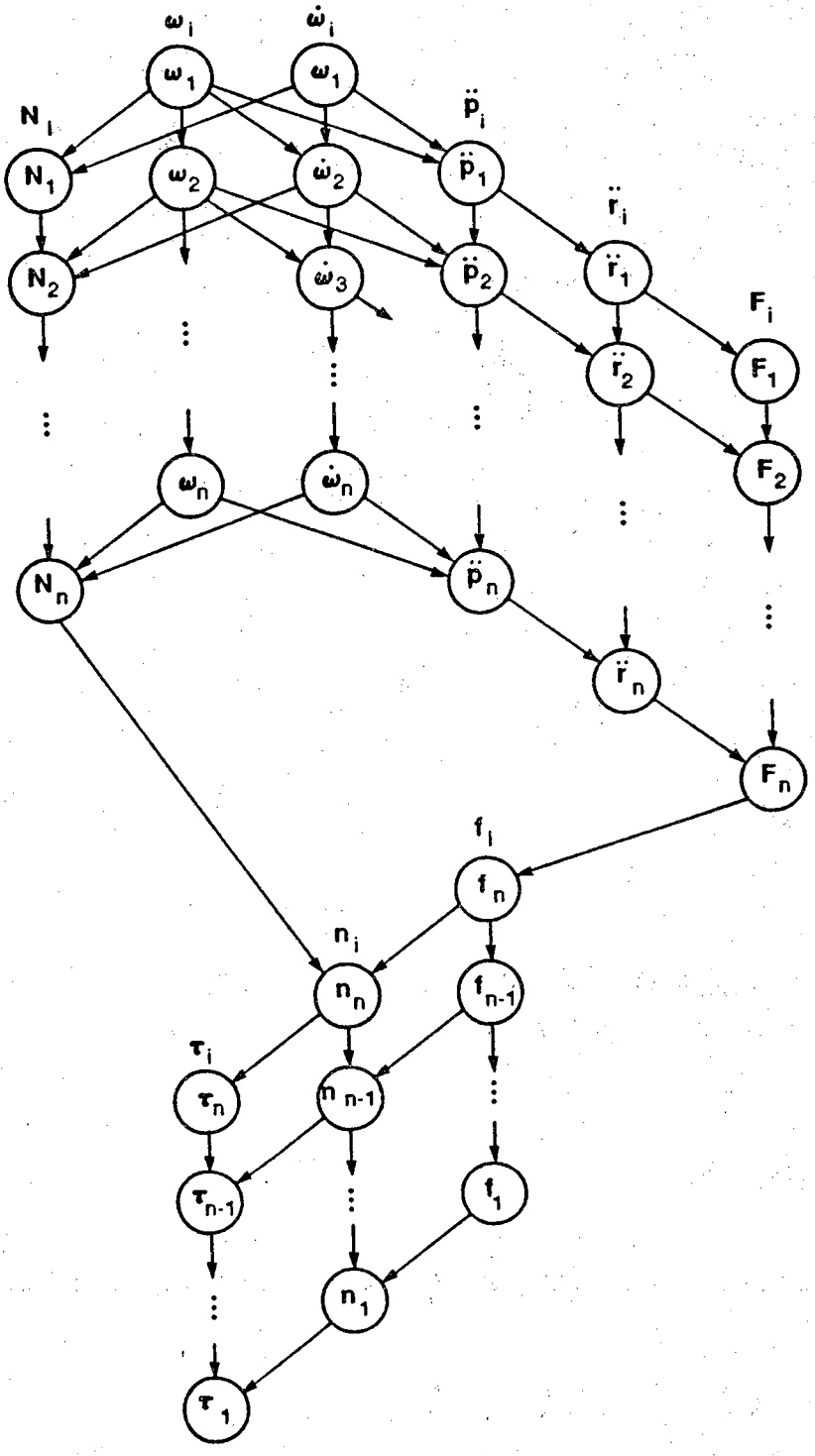


Figure 1. Linear recurrence structure of the NE equations.

Modules description

$T_1 = {}^0\mathbf{R}_i$

$T_2 = \mathbf{z}_{i-1}\dot{q}_i\,(1-\lambda_i)$

$T_3 = \omega_i$

$T_4 = \mathbf{p}_i^*$ and $\mathbf{s}_i$

$T_5 = (\mathbf{z}_{i-1}\,\ddot{q}_i + \omega_{i-1} \times \mathbf{z}_{i-1}\,\dot{q}_i)\,(1-\lambda_i)$

$T_6 = \dot{\omega}_i$

$T_7 = \omega_i \times (\omega_i \times \mathbf{p}_i^*)$

$T_8 = (\mathbf{z}_{i-1}\,\ddot{q}_i + 2\,\omega_i \times (\mathbf{z}_{i-1}\,\dot{q}_i))\,\lambda_i$

$T_9 = \dot{\omega}_i \times \mathbf{p}_i^*$

$T_{10} = T_7 + T_8 + T_9$

$T_{11} = \ddot{\mathbf{p}}_i$

$T_{12} = \omega_i \times (\omega_i \times \mathbf{s}_i)$

$T_{13} = \dot{\omega}_i \times \mathbf{s}_i$

$T_{14} = T_{12} + T_{13}$

$T_{15} = T_{11} + T_{14}$

$T_{16} = \mathbf{F}_i$

$T_{17} = \mathbf{f}_i$

$T_{18} = \mathbf{s}_i \times \mathbf{F}_i$

$T_{19} = \mathbf{p}_i^* \times \mathbf{f}_i$

$T_{20} = {}^i\omega_i$

$T_{21} = \mathbf{J}_i\,{}^i\omega_i$

$T_{22} = {}^i\dot{\omega}_i$

$T_{23} = T_{20} \times T_{21}$

$T_{24} = \mathbf{J}_i\,{}^i\dot{\omega}_i$

$T_{25} = {}^i\mathbf{N}_i$

$T_{26} = \mathbf{N}_i$

$T_{27} = T_{19} + T_{26}$

$T_{28} = T_{18} + T_{27}$

$T_{29} = \mathbf{n}_i$

$T_{30} = \tau_i$

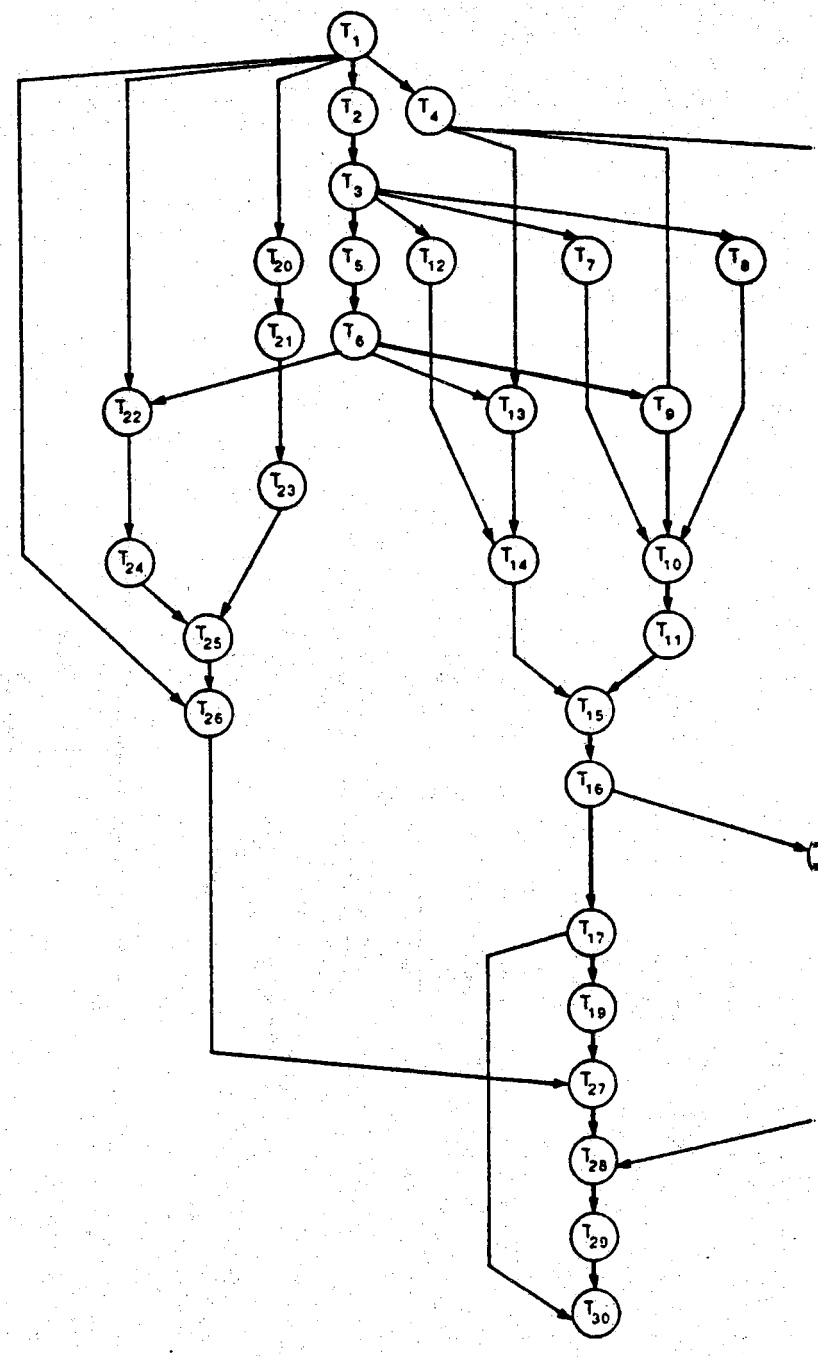


Figure 2. The NE task graphs in the HLR form.

Modules description

$T_1 = {}^{i}\mathbf{R}_{i-1}\mathbf{z}_0 q_i (1-\lambda_i)$

$T_2 = \omega_i$

$T_3 = {}^{i}\mathbf{R}_{i-1}\omega_{i-1} \times \mathbf{z}_0 \dot{q}_i (1-\lambda_i)$

$T_4 = {}^{i}\mathbf{R}_{i-1}\mathbf{z}_0 \ddot{q}_i (1-\lambda_i)$

$T_5 = T_3 + T_4$

$T_6 = \dot{\omega}_i$

$T_7 = \dot{\omega}_i \times \mathbf{p}_i^*$

$T_8 = \omega_i \times (\omega_i \times \mathbf{p}_i^*)$

$T_9 = 2\omega_i \times {}^{i}\mathbf{R}_i \mathbf{z}_0 \dot{q}_i \lambda_i$

$T_{10} = {}^{i}\mathbf{R}_{i-1}\mathbf{z}_0 \ddot{q}_i \lambda_i + T_9$

$T_{11} = T_7 + T_{10}$

$T_{12} = T_8 + T_{11}$

$T_{13} = \ddot{\mathbf{p}}_i$

$T_{14} = \omega_i \times (\omega_i \times \mathbf{s}_i)$

$T_{15} = \dot{\omega}_i \times \mathbf{s}_i$

$T_{16} = T_{14} + T_{15}$

$T_{17} = T_{13} + T_{16}$

$T_{18} = \mathbf{F}_i$

$T_{19} = \mathbf{f}_i$

$T_{20} = \mathbf{J}_i {}^{i}\omega_i$

$T_{21} = \omega_i \times T_{20}$

$T_{22} = \mathbf{J}_i \dot{\omega}_i$

$T_{23} = T_{21} + T_{22}$

$T_{24} = \mathbf{s}_i + \mathbf{p}_i^*$

$T_{25} = T_{24} \times T_{18}$

$T_{26} = T_{23} + T_{25}$

$T_{27} = \mathbf{p}_i^* \times \mathbf{f}_{i+1}$

$T_{28} = T_{26} + T_{27}$

$T_{29} = \mathbf{n}_i$

$T_{30} = \tau_i$

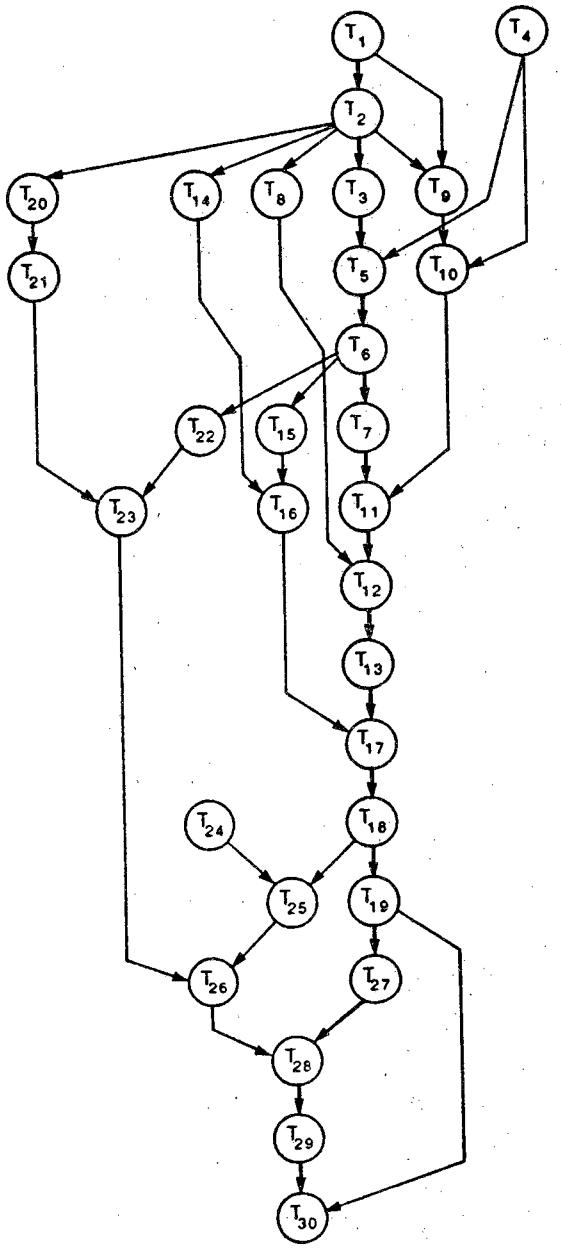
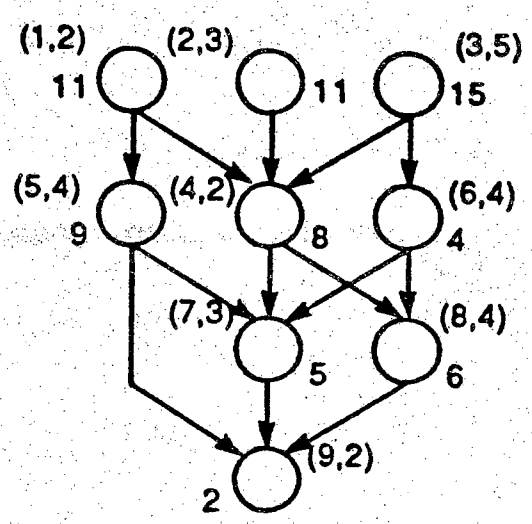


Figure 3. The NE task graphs in the IHLR form.



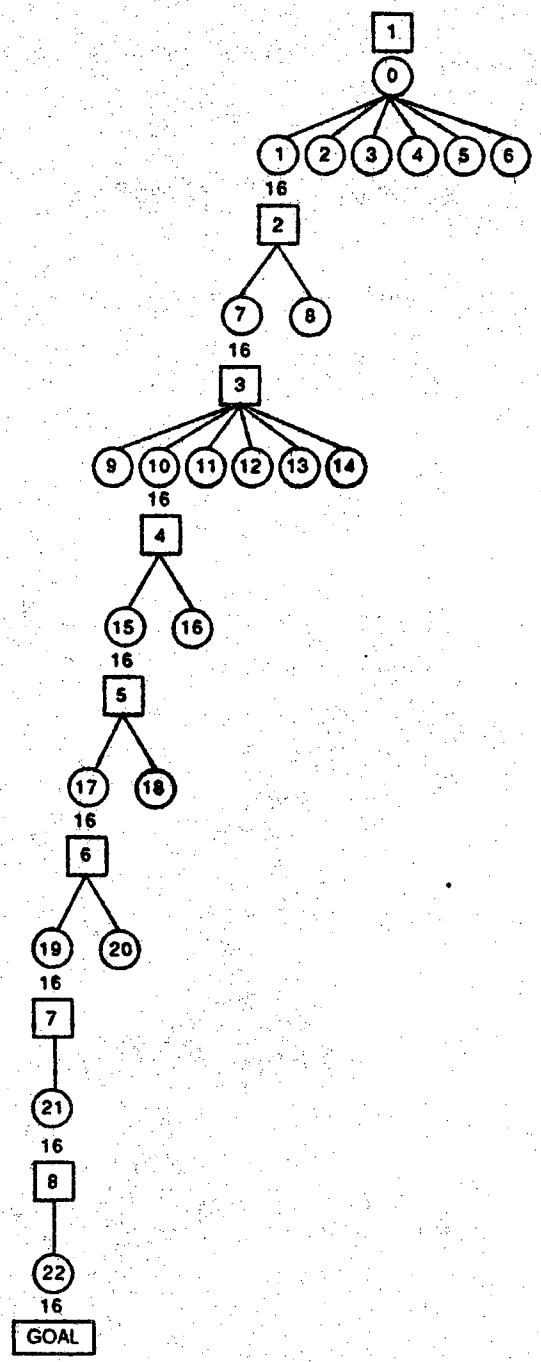
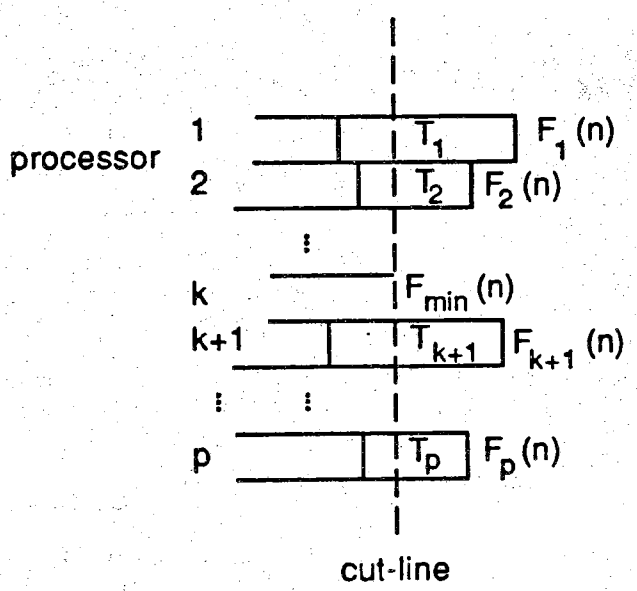


Figure 4. (a) A DATG with 9 modules. (b) The node expansion.



$$K(n) = \{(T_1, D_1), (T_2, D_2) \ldots (T_{k-1}, D_{k-1}), (T_{k+1}, D_{k+1}), \ldots (T_p, D_p)\}$$

Figure 5. The Gantt chart for the set $K(n)$.

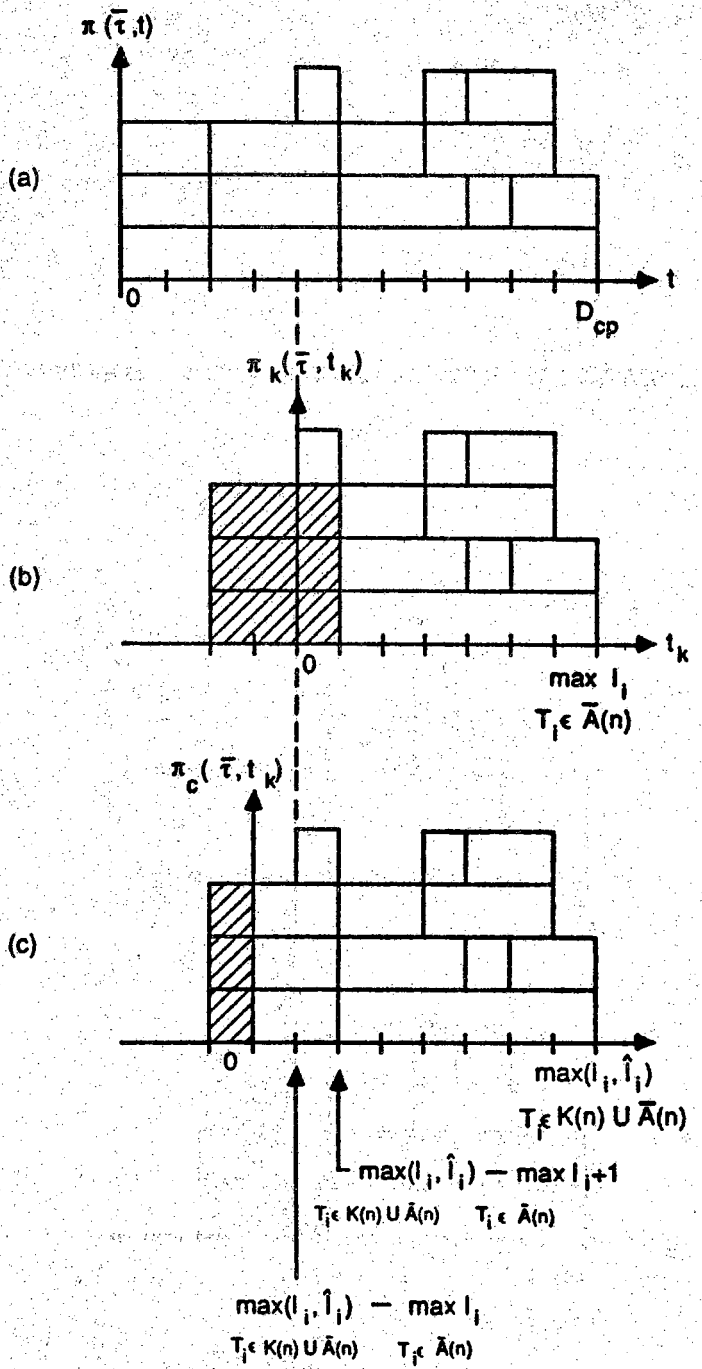
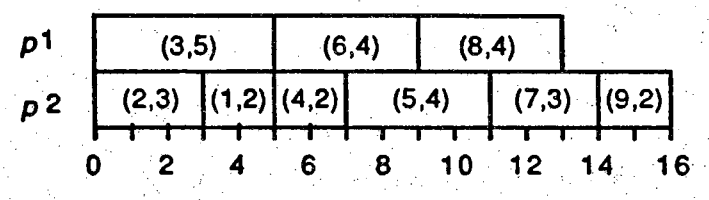
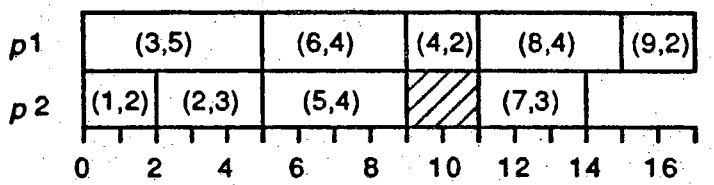


Figure 6. (a) The latest vertex activity function $\pi(\bar{\tau}, t)$ of a given interval $[0, D_{cp}]$. (b) and (c) represent Eqs. (16) and (11), respectively. The shaded areas in (b) and (c) represent the modules $T_j \notin \bar{A}(n)$ and $T_j \notin K(n) \bigcup \bar{A}(n)$, respectively. The value $t_k = 0$ in (b) corresponds to the value $t_k = \max_{T_j \in K(n) \bigcup \bar{A}(n)} (l_j, \hat{l}_j) - \max_{T_j \in \bar{A}(n)} l_j$ in (c).



(a)



(b)

Schedule Result for $p = 2$

(a) Gantt chart for MIN_LENGTH algorithm

(b) Gantt chart for list scheduling method (where shaded area represents processor is idle).

Figure 7. The Gantt charts for optimal and suboptimal schedules for the example in Fig. 4. (a) The MIN_LENGTH algorithm. (b) The list scheduling method (where shaded area indicates the processor is idle).

Table 1 Module Execution Time and Level for NE Equations Expressed in HLR Form

Module Number	Execution Time	Level
1	$(1m+2a)\lceil\log_2 n\rceil$	$3a\lceil\log_2(n+1)\rceil+4a\lceil\log_2 n\rceil+1m\lceil\log_2 n\rceil+(6-2\lambda_i)m+(10-\lambda_i)a$
2	$(1-\lambda_i)m$	$3a\lceil\log_2(n+1)\rceil+2a\lceil\log_2 n\rceil+(6-2\lambda_i)m+(10-\lambda_i)a$
3	$1a\lceil\log_2 n\rceil$	$3a\lceil\log_2(n+1)\rceil+2a\lceil\log_2 n\rceil+(5-\lambda_i)m+(10-\lambda_i)a$
4	$1m+2a$	$3a\lceil\log_2(n+1)\rceil+6m+(11+\lambda_i)a$
5	$(1-\lambda_i)(1m+2a)$	$3a\lceil\log_2(n+1)\rceil+1a\lceil\log_2 n\rceil+(5-\lambda_i)m+(10-\lambda_i)a$
6	$1a\lceil\log_2 n\rceil$	$3a\lceil\log_2(n+1)\rceil+1a\lceil\log_2 n\rceil+4m+(8+\lambda_i)a$
7	$2m+2a$	$3a\lceil\log_2(n+1)\rceil+5m+(9+\lambda_i)a$
8	$\lambda_i(2m+2a)$	$3a\lceil\log_2(n+1)\rceil+(3+2\lambda_i)m+(7+3\lambda_i)a$
9	$1m+1a$	$3a\lceil\log_2(n+1)\rceil+4m+(8+\lambda_i)a$
10	$1a+\lambda_i(1a)$	$3a\lceil\log_2(n+1)\rceil+3m+(7+\lambda_i)a$
11	$1a(\lceil\log_2(n+1)\rceil)$	$3a\lceil\log_2(n+1)\rceil+3m+6a$
12	$2m+2a$	$2a\lceil\log_2(n+1)\rceil+4m+9a$
13	$1m+1a$	$2a\lceil\log_2(n+1)\rceil+4m+8a$
14	$1a$	$2a\lceil\log_2(n+1)\rceil+3m+7a$
15	$1a$	$2a\lceil\log_2(n+1)\rceil+3m+6a$
16	$1m$	$2a\lceil\log_2(n+1)\rceil+3m+5a$
17	$1a\lceil\log_2(n+1)\rceil$	$2a\lceil\log_2(n+1)\rceil+2m+5a$
18	$1m+1a$	$1a\lceil\log_2(n+1)\rceil+2m+4a$
19	$1m+1a$	$1a\lceil\log_2(n+1)\rceil+2m+5a$
20	$1m+2a$	$1a\lceil\log_2(n+1)\rceil+5m+12a$
21	$1m+2a$	$1a\lceil\log_2(n+1)\rceil+4m+10a$
22	$1m+2a$	$1a\lceil\log_2(n+1)\rceil+4m+11a$
23	$1m+1a$	$1a\lceil\log_2(n+1)\rceil+3m+8a$
24	$1m+2a$	$1a\lceil\log_2(n+1)\rceil+3m+9a$
25	$1a$	$1a\lceil\log_2(n+1)\rceil+2m+7a$
26	$1m+2a$	$1a\lceil\log_2(n+1)\rceil+2m+6a$
27	$1a$	$1a\lceil\log_2(n+1)\rceil+1m+4a$
28	$1a$	$1a\lceil\log_2(n+1)\rceil+1m+3a$
29	$1a\lceil\log_2(n+1)\rceil$	$1a\lceil\log_2(n+1)\rceil+1m+2a$
30	$1m+2a$	$1m+2a$

Table 2 Module Execution Time and Level for NE Equations Expressed in IHLR Form

Module Number	Execution Time	Level
1	$(1-\lambda_i)m$	$(3m+4a)\lceil\log_2(n+1)\rceil+(2m+4a)\lceil\log_2 n\rceil+(6-2\lambda_i)m+8a$
2	$(1m+2a)\lceil\log_2 n\rceil$	$(3m+4a)\lceil\log_2(n+1)\rceil+(2m+4a)\lceil\log_2 n\rceil+(5-\lambda_i)m+8a$
3	$(1-\lambda_i)(1m+1a)$	$(3m+4a)\lceil\log_2(n+1)\rceil+(1m+2a)\lceil\log_2 n\rceil+(5-\lambda_i)m+8a$
4	$(1-\lambda_i)m$	$(3m+4a)\lceil\log_2(n+1)\rceil+(1m+2a)\lceil\log_2 n\rceil+(5-\lambda_i)m+(7+\lambda_i)a$
5	$(1-\lambda_i)a$	$(3m+4a)\lceil\log_2(n+1)\rceil+(1m+2a)\lceil\log_2 n\rceil+4m+(7+\lambda_i)a$
6	$(1m+2a)\lceil\log_2 n\rceil$	$(3m+4a)\lceil\log_2(n+1)\rceil+(1m+2a)\lceil\log_2 n\rceil+4m+(6+2\lambda_i)a$
7	$1m+1a$	$(3m+4a)\lceil\log_2(n+1)\rceil+4m+(6+2\lambda_i)a$
8	$2m+2a$	$(3m+4a)\lceil\log_2(n+1)\rceil+5m+(7+\lambda_i)a$
9	$\lambda_i(2m+1a)$	$(3m+4a)\lceil\log_2(n+1)\rceil+3m+(5+6\lambda_i)a$
10	$\lambda_i a$	$(3m+4a)\lceil\log_2(n+1)\rceil+3m+(5+3\lambda_i)a$
11	$\lambda_i a$	$(3m+4a)\lceil\log_2(n+1)\rceil+3m+(5+2\lambda_i)a$
12	$\lambda_i a$	$(3m+4a)\lceil\log_2(n+1)\rceil+3m+(5+\lambda_i)a$
13	$(1m+2a)\lceil\log_2(n+1)\rceil$	$(3m+4a)\lceil\log_2(n+1)\rceil+3m+5a$
14	$2m+2a$	$(2m+4a)\lceil(\log_2(n+1))\rceil+5m+8a$
15	$1m+1a$	$(2m+4a)\lceil(\log_2(n+1))\rceil+4m+7a$
16	$1a$	$(2m+4a)\lceil(\log_2(n+1))\rceil+3m+6a$
17	$1a$	$(2m+4a)\lceil(\log_2(n+1))\rceil+3m+5a$
18	$1m$	$(2m+4a)\lceil(\log_2(n+1))\rceil+3m+4a$
19	$(1m+2a)\lceil\log_2(n+1)\rceil$	$(2m+4a)\lceil(\log_2(n+1))\rceil+2m+4a$
20	$1m+2a$	$(1m+2a)\lceil\log_2(n+1)\rceil+3m+8a$
21	$1m+1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+2m+6a$
22	$1m+2a$	$(1m+2a)\lceil\log_2(n+1)\rceil+2m+7a$
23	$1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+1m+5a$
24	$1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+2m+6a$
25	$1m+1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+2m+5a$
26	$1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+1m+4a$
27	$1m+1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+2m+4a$
28	$1a$	$(1m+2a)\lceil\log_2(n+1)\rceil+1m+3a$
29	$(1m+2a)\lceil\log_2(n+1)\rceil$	$(1m+2a)\lceil\log_2(n+1)\rceil+1m+2a$
30	$1m+2a$	$1m+2a$

Table 3 Comparison of Various Methods for Computing NE Equations of Motion

Method	Multiplications	Additions
Uicker/Kahn (original Lagrangian)	$32\frac{1}{2}n^4+86\frac{5}{12}n^3+171\frac{1}{4}n^2$ $+53\frac{1}{3}n-128$ (66,271)	$25n^4+66\frac{1}{3}n^3+129\frac{1}{2}n^2$ $+42\frac{1}{3}n-96$ (51,548)
Luh, Walker, and Paul (Newton-Euler)	$150n-48$ (852)	$131n-48$ (738)
Luh and Lin (Scheduled Parallel N.E.)	$57n-18$ (323)	$50n-18$ (280)
Lee and Chang (Parallel Computation)	$27\lceil\log_2 n\rceil+116$ (197)	$24\lceil\log_2 n\rceil+9\lceil\log_2(n+1)\rceil+84$ (183)
Lathrop (Linear Parallel)	$2n+3$ (15)	$6n+7$ (43)
Lathrop (Logarithmic)	$2\lceil\log_2(n+1)\rceil+5$ (11)	$6\lceil\log_2(n+1)\rceil+4\lceil\log_2 n\rceil+8$ (28)
NE Task Graph IHLR Formulation	$3\lceil\log_2(n+1)\rceil+2\lceil\log_2 n\rceil+6-2\lambda_i$ (21)	$6\lceil\log_2(n+1)\rceil+4\lceil\log_2 n\rceil+8$ (38)
NE Task Graph HLR Formulation	$\lceil\log_2 n\rceil+6-2\lambda_i$ (9)	$3\lceil\log_2(n+1)\rceil+4\lceil\log_2 n\rceil+10-\lambda_i$ (31)

Numbers inside the parenthesis indicates the number of mathematical operations when $n = 6$.

Table 4. Cost Value of Each Node in the Search Tree in Figure 4b.

Node n	$U(n)$	$\lceil q_{ec} \rceil$	$f(n)$	$R(n)$
1	<(3,5)/5, (1,2)/2>	1	16	{(2,3), (5,4)}
2	<(3,5)/5, (2,3)/3>			
3	<(1,2)/2, (2,3)/3>			
4	<(3,5)/5, (∅,x)/5>			
5	<(2,3)/3, (∅,x)/3>			
6	<(1,2)/2, (∅,x)/2>			
7	<(3,5)/5, (2,3)/5>	1	16	{(6,4), (5,4), (4,2)}
8	<(3,5)/5, (∅,x)/5>			
9	<(6,4)/9, (5,4)/9>	0	17	
10	<(6,4)/9, (4,2)/7>	0	16	{(5,4)}
11	<(5,4)/9, (4,2)/7>			
12	<(6,4)/9, (∅,x)/9>			
13	<(5,4)/9, (∅,x)/9>			
14	<(4,2)/7, (∅,x)/7>			
15	<(6,4)/9, (5,4)/11>	0	16	{(8,4)}
16	<(6,4)/9, (∅,x)/9>			
17	<(8,4)/13, (5,4)/11>	0	16	{(7,3)}
18	<(∅,x)/11, (5,4)/11>			
19	<(8,4)/13, (7,3)/14>	0	16	{(∅,x)}
20	<(8,4)/13, (∅,x)/13>			
21	<(∅,x)/14, (7,3)/14>	0	16	{(9,2)}
22	<(∅,x)/16, (9,2)/16>	0	16	

Table 5. Optimal Modules Allocation in Each of the Six Processors for a Stanford Arm

Processor Number	Modules Number
1	1,3,21,35,45,54,44,67,14,29
2	2,7,16,15,18,20,41,55,24,60,58,33,42,39,77,87
3	4,6,9,26,31,40,50,57,61,63,64,65,66,69,71
4	5,10,27,36,53,47,28,52,22,38,82,75,76
5	17,11,37,46,48,56,68,72,73,80,81
6	8,12,19,30,34,25,43,51,59,32,49,70,23,13,28,74 79,85,78,83,84,86,88

Table 6 Comparison of Processing Time (*ms*)

Number of Processors	Luh & Lin	Kasahara & Narita/ Relative Error	MIN_LENGTH Algorithm/ Relative Error
1	24.8	$24.83/\epsilon = 0$	$24.83/\epsilon = 0$ (optimal)
2		$12.42/\epsilon = 0$	$12.42/\epsilon = 0$
3		$8.43/\epsilon = 0$	$8.44/\epsilon = 0$
4		$6.59/\epsilon \leq 0.01$	$6.59/\epsilon \leq 0.01$
5		$5.86/\epsilon \leq 0.03$	$5.72/\epsilon \leq 0.005$
6	9.67	$5.73/\epsilon \leq 0.01$	$5.70/\epsilon = 0$†
7	N/A‡	$5.70/\epsilon = 0$	N/A

† indicates the critical-path-length computation.

‡ indicates Not Applicable.

Table 7. The Optimal Schedules for Any Number of MPs for an n-Link Manipulator

Number of Modular Processors	Processing Time (μs) MIN_LENGTH	*DHLF/MISF*	Relative Error ϵ
1	800.8	800.8	0
2†	400.4	400.4	0
10	80.1	82.0	0.0237
20	41.5	43.6	0.0506
30	28.7	31.8	0.1080
38	24.2‡	25.8	0.0662

† equivalent to using six MC68020 microprocessors.

‡ indicates the critical-path-length computation.