Liu, J.F., and Abdel-Malek, K., (1999), "On the Problem of Scheduling Parallel Computations of Multibody Dynamic Analysis", *ASME Journal of Dynamic Systems, Measurement and Control*, Vol. 121, No. 3, pp. 370-376.

On the Problem of Scheduling Parallel Computations of Multibody Dynamic Analysis

J. F. Liu
Research Assistant

K. A. Abdel-Malek
Assistant Professor
Mem. ASME

Department of Mechanical Engineering
Center for Computer Aided Design
The University of Iowa
Iowa City, IA 52242

Abstract

A formulation of a graph problem for scheduling parallel computations of multibody dynamic analysis is presented. The complexity of scheduling parallel computations for a multibody dynamic analysis is studied. The problem of finding a shortest critical branch spanning tree is described and transformed to a minimum radius spanning tree, which is solved by an algorithm of polynomial complexity. The problems of shortest critical branch minimum weight spanning tree (SCBMWST) and the minimum weight shortest critical branch spanning tree (MWSCBST) are also presented. Both problems are shown to be NP-hard by proving that the bounded critical branch bounded weight spanning tree (BCBBWST) problem is NP-complete. It is also shown that the minimum computational cost spanning tree (MCCST) is at least as hard as SCBMWST or MWSCBST problems, hence itself an NP-hard problem. A heuristic approach to solving these problems is developed and implemented, and simulation results are discussed.

1 Introduction

A rigid multibody mechanism is a collection of two or more rigid bodies and zero or more rigid joints for which each joint connects exactly two distinct bodies. Every two bodies are connected by at most one joint. A connected rigid multibody mechanism is one that is impossible to partition into two disconnected parts without addition or deletion of a body or a joint. The types of joints treated in this paper are revolute, translational, cylindrical, spherical, and universal. Bae and Haug (1987a and 1987b) and Haug (1995) used the variational-vector calculus approach and the cut joint method to derive a recursive Newton-Euler formulation for constrained mechanical systems with open and closed loops. Wittenburg (1978) presented a method to handle closed loop systems by cutting joints to form a spanning tree model that has no closed loops, with Lagrange multipliers

introduced to account for cut-joint constraints. Wittenberg and Wolz (1985) presented a cut-body method to treat closed loop systems.

Since the connectivity of a mechanical system can be represented by a graph, graph theory can be used to identify the topological structure of multibody dynamics. Wittenburg (1977) and Sheth (1972) implemented graph theory to analyze the topology of multibody systems in terms of relative coordinates. Roberson (1984) presented a method to create the path matrix of a graph. Christofedes (1986) developed an algorithm to generate all spanning trees for a system graph. Kim (1984) proposed a method to evaluate weighting factors of some general kinematic joints, for the purpose of minimizing the number of generalized equations. Bae and Haug (1986) proposed another method to evaluate weighting factors, for the purpose of minimizing the chain length and maximizing efficiency in parallel computation. Zeid (1989) and more recently, Zeid and Overholt (1995) have used bond graph theory for the simulation of multibody dynamics.

The problem of parallel computation scheduling of multibody dynamic analysis has been addressed by Hwang and Haug (1989). Graph theory was used in the past (Tsai and Haug 1989) to define a body as a node and a kinematic joint as an edge. Using this representation, a closed loop is one that has the same ending and beginning. An independent loop is one that has at least one edge not appearing in other loops in the same connected graph. If there are no connected loops in the structure, the system is said to have a tree structure. If a graph is not a tree, an edge is cut in each independent closed loop to form a tree structure, called a spanning tree (the joint is called a cut joint). For example, the mechanism shown in Fig. 1a can be represented by the connected graph shown in Fig. 1b which contains two independent loops.

Fig. 1 (a) Governor mechanism (b) A connected graph with two independent loops

This method is necessary for the dynamic recursive formulation used in this paper that employs relative coordinates. This dynamics formulation is currently being used in the National Advanced Driving Simulator (NADS 1995) at the University of Iowa. The formulation is based on the following steps.

(S1) Perform the forward kinematic and backward dynamic computations. Start from the base body, determine the position and velocity of each body by traversing upward through the leaves along each branch in the graph. Start from each leave, determine the generalized force of each body by traversing downward to the base body along each branch.

(S2) Solve the equations of motion. Compute the acceleration of the base body and the second derivative of the joint coordinates by solving the equations of motion.

(S3) Integrate the acceleration of the base body and the second derivative of the joint coordinates. Repeat steps one through three again.

Since different cut joints will result in different spanning tree structures, some form of spanning tree that minimizes computation time for each step in the simulation should be determined. In order to find the best spanning tree, weighting factors must be defined for each edge. This is done so that a joint that has a large weighting is given cutting priority over a joint that has a smaller weighting, if they are in the same independent closed loop. For example, a revolute joint has one relative degree-of-freedom, while a spherical joint has three. Cutting a revolute joint eliminates one relative coordinate and introduces five constraint equations, whereas cutting a spherical joint eliminates three relative coordinates and introduces three constraint equations.

2 Computational Cost

In terms of computational speed for a parallel code implementation, the length of a critical branch of the spanning tree determines the time required to perform computations in (S1), and the total weight of all the edges in the spanning tree (proportional to the size of the resulting equations of motion) determines computation time required in (S2).

Individually minimizing the computational cost in (S1) is equivalent to finding a shortest critical branch spanning tree (SCBST). Minimizing the computational cost in (S2 and S3) is equivalent to finding a minimum weight spanning tree (MWST). To minimize the total computational cost in one iteration interval (i.e. the cost to compute steps 1-3) for a parallel-implemented dynamic analysis code, the sum of the computation cost in step 1, 2, and 3 should be minimized. This problem is formulated as follows:

Problem 1: **Minimum Computational Cost Spanning Tree (MCCST)**

Given a connected bigraph *G(V,E),* where *V* is the set of all the vertices and *E* is the set of all the edges *e,* and a weight assignment $w(e) \in Z+$ for all $e \in E$, it is necessary to determine a spanning tree $T(V, E_T)$, to minimize a cost function f_c defined by

$$f_c \equiv \ell(B_c(r,T)) + \beta \sum_{e \in E_T} w(e) \qquad (1)$$

where E_T is the set of the edges in *E* after cutting, and $\ell(B_c(r,T))$ is the length of the critical branch $B_c(r,T)$ of the spanning tree *T*, *r* is the base vertex (root) of *T*, $\sum_{e \in E_T} w(e)$ is the sum of the weight of all the edges *e* in a tree *T*, and β is a strictly positive real number which depends on the machine (computer) parameters and the actual coding.

The computational cost C is proportional to the computation time required to complete the dynamic analysis of one time interval. The total number of generalized equations of motion $N(T)$ can be computed as

$$N(T) = \sum_{e \in E'} f(e) + \sum_{e \in E - E'} \bar{f}(e) \tag{2}$$

where f is the number of joint coordinates associated with a joint, and $\bar{f}$ is the number of cut joint constraints introduced. The values for f and $\bar{f}$ for different types of joints are presented in table (1).

Joint Type	f	$\bar{f}$
Revolute joint	1	5
Translational joint	1	5
Universal joint	2	4
Cylindrical joint	2	4
Spherical joint	3	3

To compute the cost C for a sequential processor, it is necessary to compute the number of generalized equations of motion. The computational cost for a sequential processor depends upon the computational cost for (S1). This can be written as

$$C(S1) = O\left(\left|E'\right|\right) \tag{3}$$

where E' is the set of edges after forming the tree, and O denotes the order of magnitude. Similarly, the cost for (S2) and (S3) above can be written as

$$C(S2) = O\left(\left|N(T)\right|^P\right) \qquad 2 < P < 3 \tag{4}$$

$$C(S3) = O\left(\left|N(T)\right|\right) \tag{5}$$

and the total computational cost for the three steps is

$$C(S1) + C(S2) + C(S3) = O\left(\left|N(E')\right|\right) + O\left(\left|N(T)\right|^P\right) + O\left(\left|N(T)\right|\right) \tag{6}$$

The objective is to find a spanning tree $T(V, E')$, and a root r of $T(V, E')$ such that the computation cost is minimized.

Theorem 1

(a) A sequential processor $C(S1+S2+S3)$ is minimized if and only if $\left|N(T)\right|$ is minimized. Furthermore, $C(S1+S2+S3)$ can be shown to be independent of the choice of a vertex as a root.

(b) *The sufficient condition of the minimum cost of a sequential processor* can be written such that $N(T)$ is minimized if T is a minimum spanning tree.

Proof 1

(a) A choice of a root only affects $C(S1)$. However, since

$$\left|E'\right| = \left|V\right| - 1 \tag{7}$$

is a constant, then $C(S1)$ is a constant for a sequential processor. Also, since

$$C(S2+S3) = O\left(\left|N\right|^P\right) + O\left(\left|N\right|\right) \tag{8}$$

is strictly increasing as N increases, then $C(S1+S2+S3)$ is minimized if and only if $\left|N(T)\right|$ is minimized.

(b) Equation (2) is rewritten as

$$N(T) = \sum_{e \in E'} f(e) - \sum_{e \in E - E'} f(e) + 6\left|E - E'\right| \tag{9}$$

.

and $$N(T) \geq \min_{T} \sum_{e \in E'} f(e) + \max_{T} \sum_{e \in E-E'} f(e) + 6\left|E - E'\right| \tag{10}$$

Thus, if T is a minimum spanning tree, i.e., the sum of weights $w(e)$ of T is minimum, the equality of Eq. (9) holds since $w(e)$ was defined as $w(e) = f(e)$.

Computational cost for a parallel code is generally more difficult to define due to the variety of parallelisms. Nevertheless, a minimum cost theorem for a parallel processor will be presented as well as a tentative scheme to achieve the minimum cost objective.

Let w be a vertex in V such that its degree of incidence d in T is one. The set of leaves $L(T)$ is a collection of all leaves in T such that

$$L(T) = \{v | d(v) = 1 \text{ in } T\} \tag{11}$$

A branch denoted by $B(v, w, T)$ is a path from v in V to w in $L(T)$. The length of an edge e in E' is denoted by $\ell(e)$. *Note that the magnitude of* $\ell(e)$ *corresponds to the computational difficulty, while* $w(e)$ *represents the number of joint coordinates associated with* e. The length of a branch $\ell(B)$ is defined as the sum of the lengths of all edges in $B(v, w, t)$. A critical branch $B_c(v, t)$ is the longest branch in T for all possible branches starting from v in V such that

$$\ell(B_c(v, T)) = \max_{w \in L(T)} \ell(B(v, w, T)) \tag{12}$$

The center of a spanning tree r is a vertex in V such that

$$\ell(B_c(r, T)) = \min_{v \in V} \ell(B_c(v, T)) \tag{13}$$

The set of centers of a spanning tree $R(T)$ is a collection of all r in T. The center of shortest critical branch spanning tree r_s is a vertex in V such that

$$\ell(B_c(r_s, T_s)) = \min_{T} \left[\min_{v \in V} \ell(B_c(v, T)) \right] \tag{14}$$

The shortest critical branch spanning tree T_s is the spanning tree where r_s resides. Define a computational cost for a parallel processor as

$$C(S1) = O[\ell(B_c(r, T))] \tag{15}$$

$$C(S2) = O(\left|N(T)\right|^P); \qquad 2 < P < 3 \tag{16}$$

$$C(S3) = O(1) \tag{17}$$

and the total cost is

$$C(S1 + S2 + S3) = C(S1) + C(S2) + C(S3) \tag{18}$$

3 Choosing a Candidate for the Minimum Cost Spanning Tree

To minimize the total computational cost in S1-S3, the MCCST problem need be addressed. Let T_m be the minimum spanning tree and T_s be the critical branch spanning tree of a graph $G(V, E)$, such that $r_s \in R(T_s)$ and $r_m \in R(T_m)$.

Theorem 2

A necessary condition for the minimum cost of a parallel processor can be written such that if T^* and r^* yield the minimum cost, then the following inequality holds

$$\ell(B_c(r_s, T_s)) \leq \ell(B_c(r^*, T^*)) \leq \ell(B_c(r_m, T_m)) \tag{19}$$

and similarly for the total number of generalized equations of motion

$$N(T_m) \le N(T^*) \le N(T_s) \tag{20}$$

Proof 2

(a) $$N(T_m) \le N(T^*) \text{ and } \ell(B_c(r_s, T_s)) \le \ell(B_c(r^*, T^*)) \tag{21}$$

(b) If $\ell(B_c(r_m, T_m)) \le \ell(B_c(r^*, T^*))$ then T_m and r_m have smaller cost than T^* and r^*, which is a contradiction.

(c) If $N(T_s) \le N(T^*)$ then T_s and r_s have smaller cost than T^* and r^*, which also contradicts the assumption.

An algorithm for determining the center of a given spanning tree has been implemented but will not be discussed in this paper.

4 Minimum Radius Spanning Tree (MRST) and Shortest Critical Branch Spanning Tree (SCBST)

Since minimizing computations in performing forward kinematics and backward dynamics is equivalent to finding a SCBST and since minimizing computations in solving the equations of motion and integrating is equivalent to finding a MWST, it is necessary to discuss a solution. In this section, a polynomial algorithm is presented to solve both MRST and SCBST problems.

4.1 Construction of MRST for a Given Graph.

Theorem 3

Let $P_s(u,v)$ be the shortest path connecting *u* and *v* and let $P_s(u,v)$ pass through w. Let $P_s(u,v)$ be divided into two parts: P_1 and P_2, such that

$$P_s(u,v) \rightarrow P_1(u,w) + P_2(w,v) \tag{22}$$

then $P_s(u,w) = P_1(u,w)$ and $P_s(w,v) = P_2(w,v)$.

Proof 3

If the contrary is assumed, it is then possible to find $P_s(u,w)$ and $P_s(w,v)$ such that

$$\ell\big(P_s(u,w)\big) < \ell\big(P_1(u,w)\big) \tag{23}$$

or

$$\ell\big(P_s(w,v)\big) < \ell\big(P_2(w,v)\big) \tag{24}$$

In the case of Eq. (23), then

$$\ell\big(P_s(u,w) + P_2(w,v)\big) < \ell\big(P_1(u,w) + P_2(w,v)\big) = \ell\big((P_s(u,v)\big) \tag{25}$$

which is a contradiction. A contradiction can also be shown for the case of Eq. (24).

The above theorem guarantees that a shortest path spanning tree (SPST) for a given graph with respect to a specific vertex u[V can be constructed as follows.

Algorithm A1: **Construction of the shortest path spanning tree (SPST) with respect to a specific vertex u in V**

F1: copy V to V'

F2: pick a specific vertex v from V'

f2.1: find the shortest path *P*(*u*,*v*) connecting u and v (Dijkstra's algorithm)

f2.2: record the predecessor of each vertex in path P

f2.3: record the distance from each vertex in path P to u

f2.4: remove all the vertices in path P from V'

F3: repeat F2 until V' is empty

The minimum radius spanning tree (MRST) for a given graph is found by (1) constructing the SPST for every vertex of the graph by applying algorithm A1 and (2) recording the longest distance appearing in each SPST. The SPST that has the minimum longest distance among all SPST's is the MRST for the given graph.

Algorithm A1 shows the existence of a shortest path spanning tree with respect to an arbitrary vertex in the graph, and that the length of the longest path in SPST equals the length of the longest shortest path in the original graph. Furthermore, it is shown that the MRST for a given graph is one of the SPSTs. These results will be used to prove the equivalence of the MRST and SCBST.

An algorithm proposed by Pape (1980) for finding shortest path from a specific vertex to all other vertices in a given graph has complexity of $\left|V^2\right|$. Therefore, the complexity of finding MRST is reduced to $\left|V^3\right|$. This algorithm will be implemented as a subroutine called upon by the heuristic approach presented in Sec. 6.2.

4.2 Finding the Shortest Critical Branch Spanning Tree.

Minimizing the cost in computing forward kinematics and backward dynamics is equivalent to finding the SCBST.

Theorem 4

The minimum radius R_m of a given graph *G* equals the length of the critical branch of a shortest critical branch spanning tree of *G*, i.e.

$$R_m = \min_{u \in V} R(u) = \ell\left(B_c(r_s, T_s)\right) \tag{26}$$

where

$$R(u) = \max_{v \in V}\left(\min_{P \in \mathrm{P}} \ell\left(P(u,v)\right)\right) \tag{27}$$

and

$$\ell\left(B_c(r_s, T_s)\right) = \min_{T}\left(\min_{v \in V}\left(\max_{w \in L(T)} \ell(B(v,w,T))\right)\right) \tag{28}$$

Proof 4

(1) Suppose $R_m < \ell\left(B_c(r_s, T_s)\right)$ and let R^* be the minimum radius of *G* by restricting the path $P(u,v)$ to take only the edges in T_s, then $R^* = \ell\left(B_c(r_s, T_s)\right)$. This is a contradiction since R^* can not be less than $\min_{u \in V} R(u)$.

(2) Suppose $R_m < \ell\left(B_c(r_s, T_s)\right)$, a minimum radius spanning tree T^* can be constructed for *G* using *A1*. Since

$$\ell\left(B_c(r, T^*)\right) = \min_{u \in V} R(u)$$

it is again a contradiction since

$$\ell\left(B_c(r_s, T_s)\right) = \min_{T} \ell\left(B_c(r, T)\right) \le \ell\left(B_c(r, T^*)\right) \tag{29}$$

The above theorem implies that the shortest critical spanning tree of a given graph can be obtained by constructing the MRST of the graph. The specific vertex in algorithm *A1* that yields the MRST is the center of the resulting shortest critical spanning tree algorithm has been shown to be of polynomial complexity.

4.3 Example of Finding SCBST for a Given Graph. Figure 2a shows a graph of 20 vertices and 32 edges. The input to the experimental code is the neighbor and adjacent edge weight list for each vertex in the graph. The output of this program is an array of the predecessor for each vertex in the graph. Figure 2b shows the output SCBST. Note that the SCBST for a given graph may not be unique. In fact, the number of all SCBST's for a given graph can be exponential with respect to the number of vertices in the graph. This will be shown in the following section.

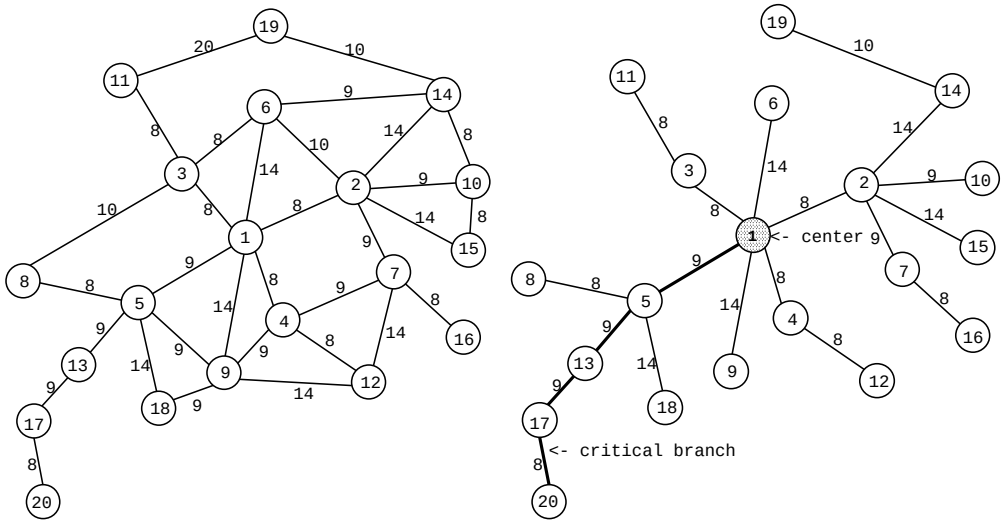

Fig. 2 (a) A Graph *G(V,E)* and (b) An SCBST of *G*

5 SCBMWST and MWSCBST Problems

5.1 Problem Formulation.

Problem 2: **Shortest Critical Branch Minimum Weight Spanning Tree (SCBMWST)**

Among all the shortest critical branch spanning trees $T_s(V, E_{T_s})$ of a given graph, it is necessary to find one such that the sum of the weight of all edges $\sum_{e \in E_{T_s}} w(e)$ is minimized.

Problem 3: **Minimum Weight Shortest Critical Branch Spanning Tree (MWSCBST)**
Among all the minimum weight spanning trees $T_m(V, E_{T_m})$ of a given tree, it is necessary to find one such that $\ell\left(B_c(r, T_m)\right)$ is minimized.

Note that if the number of SCBST's or MWST's for a given graph does not increase exponentially with the size of the problem $|E|$, then either of the above problems can be solved within polynomial time by exhaustively searching all possible SCBST's or MWST's

(Hwang and Haug 1989). It will be shown however, that polynomial algorithms for both problems do not exist if $NP \neq P$ by proving that problems 2 and 3 are NP hard.

Problem 4: **Bounded Critical Branch Bounded Weight Spanning Tree (BCBBWST)** Given a connected bigraph $G(V,E)$, a weight assignment w(e)∈Z+ for all e∈ E, and positive integers K_1 and K_2, it is necessary to find a spanning tree $T(V, E_T)$ for G such that

$$\sum_{e \in E_{T_s}} w(e) \leq K_1 \tag{30}$$

and

$$\ell\left(B_c(r,T)\right) \leq K_2 \tag{31}$$

5.2 NP-Completeness of BCBBWST Problem.

Theorem 5: **BCBBWST problem is NP-complete**.

The following proof of theorem 3 is a modification of the proof given in Ho et al. (1991), where the NP-completeness of the bounded diameter bounded cost spanning tree of a given network is proved.

Proof 5

(1) BCBBWST is NP.

(2) In the following discussion, it will be shown that a polynomial reduction from 3SAT to BCBBWST can be achieved.

(2.1) Let $C = \{C_1, C_2, ..., C_q\}$ be an instance of 3SAT over variable set $X = \{X_1, X_2, ..., X_n\}$. A graph $G(V, E)$ will be constructed and a weight function $w(e)$ will be assigned such that C is satisfied if and only if there is a spanning tree $T(V, E_T)$ for G where

$$\sum_{e \in E_T} w(e) \leq 3n + 3q + 5 \tag{32}$$

and

$$\ell\left(B_c(r,T)\right) \leq 5 \tag{33}$$

The construction of G is depicted in Fig. 3. It contains the following vertices: the true assignment node r, the variable $\{x_1, \bar{x}_1, x_2, \bar{x}_2, ..., x_n, \bar{x}_n\}$, the clause nodes $\{c_1, c_2, ..., c_q\}$, and a special node s. G also contains the following edges: assignment edges $\{(r, x_i)|$ from r to all variable nodes$\}$, contrast edges $\{(x_i, \bar{x}_i)|$ for $i = 1, ..., n\}$, clause edges $\{(u_i, c_j)|u_i = x_i$ or $\bar{x}_i\}$ depending on whether $C_j[C$ contains x_i or $\bar{x}_i$, respectively, and a special edge (r, s). The assignment edges have weight 2; contrast edges have weight 1; clause edges have weight 3; and the special edge has weight 5. *It is clearly a polynomial time-construction.*

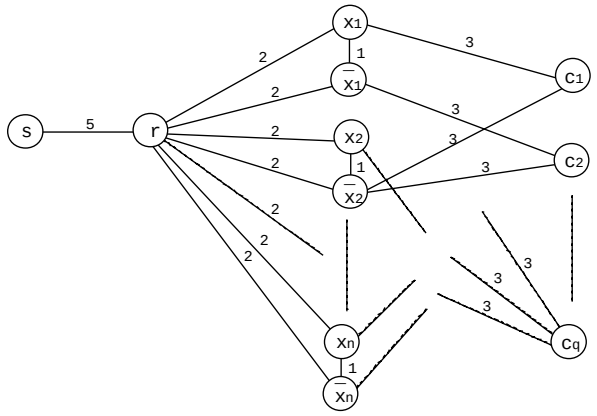

Fig. 3 Construction of *G*

(2.2) Suppose C is satisfied, then there is an assignment such that every clause in C contains at least one true variable. It may be possible that *T* consists of the following edges: (1) edge *(r,s)*, (2) (r, x_i) for all *i* if variable x_i is assigned true or $(r, \bar{x}_i)$ if x_i is assigned false, (3) all the contrast edges, and (4) the clause edges (x_j, c_i) for all *i* where x_j is the first true variable in clause C_i. Clearly, *T* is a spanning tree and

$$\sum_{e \in E_T} w(e) \le 3n + 3q + 5 \tag{34}$$

It can also be seen that *r* is the center of the graph with

$$\ell\big(B_c(r,T)\big) = 5 \tag{35}$$

(2.3) Conversely, suppose that there is a spanning tree $T(V, E_T)$ for *G* such that

$$\sum_{e \in E_T} w(e) = 3n + 3q + 5 \tag{36}$$

and

$$\ell\big(B_c(r,T)\big) \le 5 \tag{37}$$

It follows that there exists $(2n+q+1)$ edges in *T* since there are $(2n+q+2)$ vertices in *G*. The tree *T* must include the special edge *(r,s)* since *(r,s)* is the only edge connecting s. Moreover, there are at least *q* clause edges in *T*, because the clause nodes are connected only by the clause edges. Assume that the spanning tree *T* consists of (*i)* assignment edges, (*j)* contrast edges, *(k+q)* clause edges, and the edge *(r,s)*. Then the following inequalities are valid.

$$i, j, k \ge 0, \quad j \le n \tag{38}$$

$$2i + j + 3k \le 3n \tag{39}$$

$$i + j + k = 2n \tag{40}$$

Assume $j < n$. Multiplying Eq. (40) by 2 and subtracting it from Eq. (39), yields

$$k \le j - n < 0 \tag{41}$$

which is a contradiction. Therefore, *T* must include all contrast edges. Substituting *j=n* into Eqs. (40) and (39), yields

$$i+k=n \tag{42a}$$

$$2i + 3k \le 2n \tag{42b}$$

Multiplying Eq. (41) by two and subtracting it from Eq. (42) yields $k \le 0$. Therefore, *T* must contain exactly *q* clause edges, *n* contrast edges, and *n* assignment edges.

In the above discussion, it was assumed that all the clause nodes are adjacent only to those variable nodes which are adjacent to node *r* in a spanning tree *T*. In order to continue this discussion, we will assume that the above claim is not valid. Since clause nodes are only adjacent to variable nodes in graph *G*, there exists a variable node *x* and a clause node *c* such that *x* is adjacent to *c* but *x* is not adjacent to node *r* in *T*. Therefore, the branch from *c* to *r* must pass through one of the contrast edges and the length of this branch is 6, which is a contradiction to $\ell\big(B_c(r,T)\big) \le 5$.

Since *T* contains all the contrast edges, *T* must contain exactly one of the edges (r, x_i) and $(r, \overline{x}_i)$ for all *i*. Assigning 'true' to all the variables adjacent to *r* and 'false' to their complements, all the clauses in *C* are satisfied because every clause contains at least one true variable. Based upon (2.1) through (2.3), *it can be seen that 3SAT is polynomially reducible to BCBBWST.*

5.3 NP-hard Problems. Since the BCBBWST problem is the decision version of the SCBMWST and MWSCBST problems, theorem 3 states that both SCBMWST and MWSCBST problems are NP-hard. Therefore, the following theorem is evident.

Theorem 6

Given a connected bigraph *G*(*V*,*E*), a weight assignment $w(e) \in Z^+$ for all $e \in E$, The problem of finding the following spanning trees for *G* are NP-hard:

(1) Shortest critical branch minimum weight spanning tree (**SCBMWST**)
(2) Minimum weight shortest critical branch spanning tree (**MWSCBST**)
(3) Bounded critical branch minimum weight spanning tree (**BCBMWST**)
(4) Bounded weight shortest critical branch spanning tree (**BWSCBST**)
(5) Minimum computational cost spanning tree (**MCCST**)

The relative difficulties of the spanning tree problems are shown in Fig. 4.

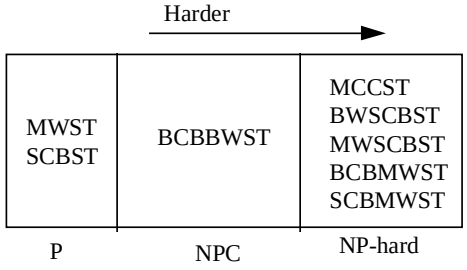

Fig. 4 Relative difficulty of spanning tree problems

6 Heuristic Algorithms for MCCST Problem

Theorem 7: **Lower Bound On the Cost Function** f_c

Let $T_s(V, E_{T_s})$ be the shortest critical branch spanning tree and $T_m(V, E_{T_m})$ be a minimum weight spanning tree of *G*(*V*,*E*), *b>0*, and f_c be defined as

$$f_c(T) \equiv \ell\big(B_c(r,T)\big) + \beta \sum_{e \in E_T} w(e) \tag{43}$$

where $T(V, E_T)$ is a spanning tree of *G*. There exists a lower bound on the cost function f_c such that

$$\min_T f_c(T) \geq \ell\left(B_c(r, T_s)\right) + \beta \sum_{e \in E_{T_m}} w(e) \tag{44}$$

Proof 7

By definition,

$$\ell\left(B_c(r, T)\right) \geq \ell\left(B_c(r, T_s)\right) \tag{45}$$

and

$$\sum_{e \in E_T} w(e) \geq \sum_{e \in E_{T_m}} w(e) \tag{46}$$

thus,

$$f_c(T) = \ell\left(B_c(r, T)\right) + \beta \sum_{e \in E_T} w(e) \geq \ell\left(B_c(r, T_s)\right) + \beta \sum_{e \in E_{T_m}} w(e) \tag{47}$$

for any $T(V, E_T)$ of *G*.

6.1 General Structures of Algorithms. The General Structure of Heuristic Algorithms is demonstrated by using the following steps.

Step 1: Start from an initial spanning tree *T(V, E_T)* of *G(V,E)*, and calculate the cost function $f_c(T)$.

Step 2: Temporarily exchange an edge e_1 of E_T with another edge e_2 of $E - E_T$. If a spanning tree T' is formed and $f_c(T') < f_c(T)$, make this exchange permanent (i.e. $T \leftarrow T'$), otherwise, revert to the previous condition.

Step 3: Determine whether the termination criteria is satisfied.

A class of algorithms can be formed based on the above structure. For example, in step 1, an SCBST or an MWST of *G* is chosen as the initial spanning tree of *G*. In step 2, different heuristic laws are applied such that e_1 and e_2 are selected. There may be more than one ending criteria that determines how the process is terminated.

6.2 Heuristic Algorithms. In this section, algorithms which start from an SCBST of *G* will be introduced. Each of the edges connecting leaves of the initial SCBST will be checked to determine whether an exchange with edges not in SCBST can reduce the cost function. Once this operation is completed, all the leaves are cut-off the spanning tree and each of the edges connecting leaves of the trimmed spanning tree is checked. The process is terminated if all the vertices in *G* are cut from the spanning tree.

6.2.1 Algorithm A2 The following nomenclature is used in discussing the algorithm.

f(i):	Predecessor of vertex *I*	$d_T(i)$:	Degree of incidence of i in *T*
p(j,i):	*j-th* neighbor of vertex *I*	$p_T(j,i)$:	Neighbor list for vertex *i* in *T*
q(j,i):	Edge weight of i and *p(j,i)*	$q_T(j,i)$:	Edge weight list for vertex *i* in *T*
r(i):	Distance from i to the center	*S:*	Stack containing vertices whose $d_T = 1$
d(i):	Degree of incidence of i in *G*	δf_c:	Change of cost function
L_{CB}:	Length of critical branch		

1: (a) Write graph *G(V,E)* in the form of *d(i),p(j,i)* and *q(j,i)*
(b) Use the experimental code of Sec. 4.3 which returns a SCBST of $T(V, E_T)$ in the form of *f(i), r(i),* $d_T(i)$, $p_T(j,i)$, $q_T(j,i)$, f_c, and L_{CB}
(c) Sort the neighbor list *p(j,i)* for each *i* by the value of *q(j,i),* neighbor with larger *q* proceeds to the bottom of the list

2: (a) Move all the leaves (vertices with $d_T = 1$) into stack *S*, vertices with smaller *r* stay on top.
(b) Stop when S has no components.

3: (a) Take the topmost vertex *v* out of *S*

(b) Discuss the vertex *v*

if *f(v)=0* (i.e. *v* is the center) then there is no need to exchange edges, stop.
let *u=f(v)* (i.e. *u* is the predecessor of *v*)
if u=*p(1,v)* (i.e. edge *u-v* has the smallest weight), there is no need to exchange edges, go to step 4
for *j=1, d(v)* (search for all the adjacent vertices of v)
{ if vertex *p(j,v)* has not been cut
{ L← *r(p(j,v))+q(j,v)* (where *L* is the candidate length of critical branch)
$\delta f_c \leftarrow b[q(j,v)-w(e(u,v))]+\max(0,L- L_{CB})$
if δf_c <0 (satisfy condition $f_c(T') < f_c(T)$, perform exchange)
{ let *w=p(j,v)*
(exchange edge *u-v* in E_T with edge *w-v* in $E - E_T$)
if $d_T(w)=1$ (*w* was originally a leaf), delete *w* from *S*
$d_T(w) \leftarrow d_T(w)+1$
$d_T(u) \leftarrow d_T(u)-1$
if d_T(u)=1 (*u* becomes a leaf), move *u* into *S*
update p_T and q_T
determine the center, update center, L_{CB}, *f* and *r*
go to 4 } } }

4: Repeat step 3 until *S* has no components.

5: Cut all the leaves in E_T, and update d_T, p_T and q_T, then start at step 2.

6.2.2 Example of Application of Algorithm A2 In the following example, let β =1 and *G* is the graph shown in Fig. 2a. The solid line in Fig. 5a shows the initial SCBST of *G*. Each leaf of the SCBST (dark vertices in Fig. 5a) is studied as in step 3 of algorithm A2. Leaves with shorter distance from the center are studied first. Therefore, vertices 6-9-11-12-10-8-18-16-19-20 are discussed successively. Edges e_{1-6}, e_{1-9}, e_{2-10}, and e_{5-18} in E_T are exchanged with edges e_{3-6}, e_{4-9}, e_{10-15}, and e_{9-18} in $E - E_T$, respectively. The resulting spanning tree for all the leaves of the initial SCBST is shown in Fig. 5b. Note

that after the exchange of edge e_{2-10} with e_{10-15}, vertex 15 is no longer a leave of the spanning tree and it is deleted from the discussion list.

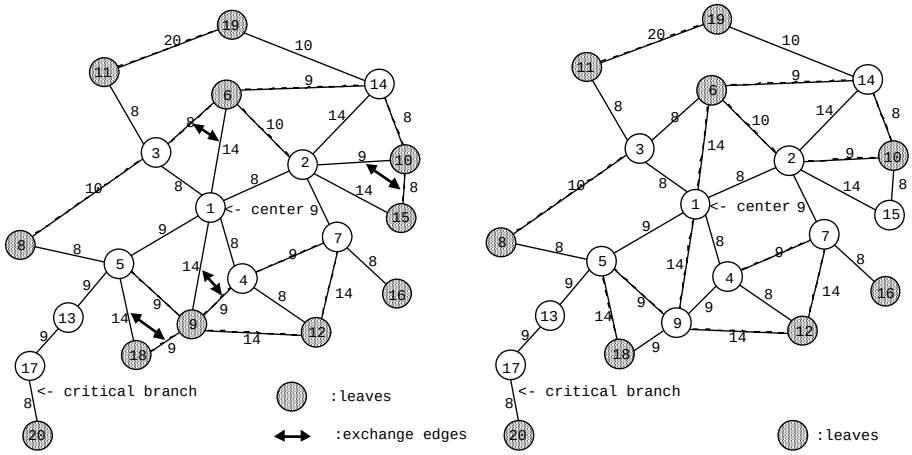

Figure 5 (a) Initial SCBST of *G* (b) the resulting tree after first iteration of vertex discussion

All the leaves of the spanning tree in Fig. 5b are cut-out. The second round of vertex discussion is then performed on each leaf of the remaining graph (Fig. 6a). This process is continued until all vertices of graph *G* are cut. During the second and successive discussions, no exchange of vertices is executed since the cost function f_c cannot be further reduced. The resulting spanning tree is shown in Fig. (6b), where $f_c = 207$.

Note that the final spanning tree determined by applying algorithm A2 is not one of the MCCST of graph *G*. By comparing the MCCST of *G* in Fig. (5a) ($f_c = 196$) with the spanning tree in Fig. (6b), it is clear that edge e_{2-14} should be exchanged with edge e_{10-14} to reduce the total weight of the spanning tree. However, because vertex 10 and its adjacent edges have been cut before the beginning of the second discussion round (Fig. 6a), the algorithm falls into a 'local minimum'.

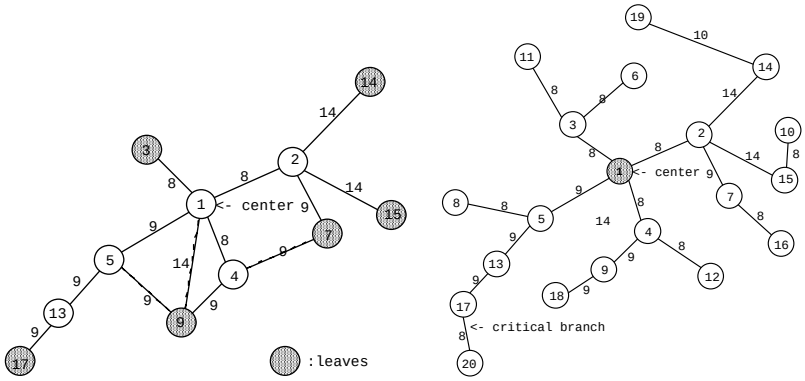

Fig. 6 (a) Cut Graph (b) The Final Spanning Tree by Algorithm I

6.2.3 Algorithm A3 In this section, a modification of one of the heuristics used in algorithm *A2* will be introduced to avoid the problem of falling into a local minimum. The heuristics in step 2 of algorithm *A2* can be changed as follows:

Move all the leaves (vertex with $d_T = 1$) into stack *S*, while vertices with smaller *r* stay on top. Steps 1, 3, 4, and 5 are kept the same as those in algorithm *A2*.

6.2.4 Example of Application of Algorithm A3 Both the test problem and the procedures are the same as those as described above. The graph after the first round of discussion is identical to that of Fig. 5b.

In the second round, vertices 17-14-7-9-3 are discussed successively, and edges e_{2-14} and e_{1-9} in E_T are exchanged with edges e_{10-14} and e_{4-9} in $E - E_T$, respectively (Fig. 8). No further chance of reduction of the cost function is detected after the second round.

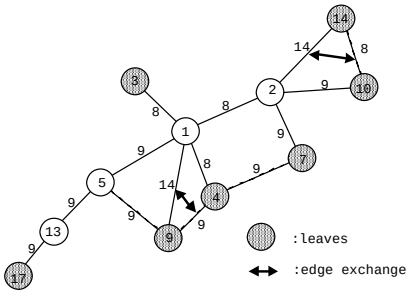

Figure 7 Second Iteration Vertex Discussion

For the test problem, the modification of the heuristic in step 3 avoids the search from proceeding into a local minimum and successfully finds a MCCST of the given graph. Note that modifications of other parts of heuristics used in algorithms A2 and A3 can also be done. For example, a search of the vertices adjacent to the center can be conducted, then proceeding towards the leaves of the spanning tree.

6.3 Complexity The complexity analysis given in the following steps is valid for both algorithms *A2* and *A3*.

Step 1 : The complexity of preprocessing (finding a SCBST) of *G* is of $|V^3|$. The complexity of sorting *p* is of $|V^2|$

Step 2 The complexity of moving leaves into S and sorting S is of $O|V^2|$.

Steps 3-4 Vertex discussion: The complexity of finding a center is of order |*V*|. Thus, the complexity of each vertex discussion is of order |*V*|. Since it is necessary to iterate until all vertices in the graph have been discussed (and cut), the total complexity is of order $|V^2|$.

Step 5 The complexity of updating d_T, p_T and q_T in this step is of order $|V^2|$

The complexity of preprocessing is of order $|V^3| + |V^2| \to |V^3|$ and the complexity of steps 2-5 is of order $|V^2| + |V^2| + |V^2| \to |V^2|$. Therefore, the complexity of algorithms A2 and A3 is of order $|V^3| + |V^2| \to |V^3|$. Note that the complexity of preprocessing (experimental code to find SCBST) is dominant in both algorithms.

7 Implementation

In this section, algorithm II will be implemented for the test problem given in Sec. 4.3 (Fig. 2a with $\beta = 1$) and for randomly generated graphs with larger number of vertices (e.g., |V|=1000).

7.1 Data Structure. The incidence array *d(i)*, neighbor list *p(j,i)*, and edge weight list *q(j,i)* are utilized to store the graph *G(V,E)*. Similarly, $d_T(i)$, $p_T(j,i)$, and $q_T(j,i)$ are used to store the current spanning tree $T(V, E_T)$. In addition, arrays $f(i)$ and $r(i)$ are used to store the predecessor and distance from the center of each vertices, respectively. Note that p, q, p_T, q_T are sparse matrices. The memory needed to store graph *G* and the current spanning tree *T* are presented in Table 2.

	size	type		size	type
d	|V|	integer	p_T	2(|V|-1)	integer
p	2|E|	integer	q_T	2(|V|-1)	real
q	2|E|	real	f	|V|	integer
d_T	|V|	integer	r	|V|	real

Total memory $\approx 8|V|+4|E|-4$.

7.2 Experimental Results. The following shows the execution of the program for the test problem depicted in Fig. 2a. The program generates the contents of stack *S* at the beginning of each round of vertex discussion. The program also generates the outcome of every vertex discussion and the change of cost function at the end of each vertex discussion. The resulting spanning tree is showed at the end of the execution of the program in the form of a predecessor list of the spanning tree. Note that the execution of the program exactly matches the desired process illustrated in Sec. 6.2.2.

```
initial fc= 224
beginning of  1 round
 STACK= 20 19 16 18 9 15 8 11 12 10 6
 discuss vertex 20  -> no edge exchange can reduce fc
 discuss vertex 19  -> no edge exchange can reduce fc
 discuss vertex 16  -> no edge exchange can reduce fc
 discuss vertex 18  -> exchange edge  5-18 in T with edge  9-18
not in T
              -> dfc= -5
              -> center= vertex 1
              -> delete vertex  9 from STACK
 discuss vertex 15  -> exchange edge  2-15 in T with edge 10-
15 not in T
              -> dfc= -6
              -> center= vertex 1
              -> delete vertex 10 from STACK
 discuss vertex  8  -> no edge exchange can reduce fc
 discuss vertex 11  -> no edge exchange can reduce fc
 discuss vertex 12  -> no edge exchange can reduce fc
 discuss vertex 6  -> exchange edge  1- 6 in T with edge  3- 6
not in T
              -> dfc= -6
              -> center= vertex 1
beginning of  2 round
 STACK= 17 14 10 7 9 4 3
 discuss vertex 17  -> no edge exchange can reduce fc
 discuss vertex 14  -> exchange edge  2-14 in T with edge 10-
14 not in T
              -> dfc= -6
              -> center= vertex 1
              -> delete vertex 10 from STACK
 discuss vertex  7  -> no edge exchange can reduce fc
 discuss vertex  9  -> exchange edge  1- 9 in T with edge  4- 9
not in T
              -> dfc= -5
              -> center= vertex 1
              -> delete vertex  4 from STACK
 discuss vertex  3  -> no edge exchange can reduce fc
beginning of  3 round
 STACK= 13 10 4
 discuss vertex 13  -> no edge exchange can reduce fc
 discuss vertex 10  -> no edge exchange can reduce fc
 discuss vertex  4  -> no edge exchange can reduce fc
beginning of  4 round
 STACK= 5 2
 discuss vertex 5  -> no edge exchange can reduce fc
 discuss vertex 2  -> no edge exchange can reduce fc
 STACK IS EMPTY! EXECUTION COMPLETED!!
```

Resulting Spanning Tree: (vertex number->predecessor number)

1 -> 0
2 -> 1
3 -> 1
4 -> 1
5 -> 1
6 -> 3
7 -> 2
8 -> 5
9 -> 4
10 -> 2
11 -> 3
12 -> 4
13 -> 5
14 -> 10
15 -> 10
16 -> 7
17 -> 13
18 -> 9
19 -> 14
20 -> 17

Total Cost Function Reduction = 28
Final fc = 196

CPU-time for different sizes of the given graphs are presented in Table 3.

Table 3 CPU-time

# vertices	Center	$\ell(B)$	$\sum w(e)$	f_c	
1000	5	45.0	3196.0	3641	0.2
2000	4	51.0	6289.0	6340	1.7
3000	1	53.0	9042.0	9095	4.8
4000	3	56.0	11245.0	11301	9.9

The CPU time is proportional to $|V^3|$.

8 Conclusions

A minimum computational cost spanning tree (MCCST) problem corresponding to the computational scheduling for parallel multibody dynamic analysis code implementation has been formulated. The cost of computing forward kinematics and backward dynamics was posed as finding the shortest critical branch spanning tree (SCBST). The problem of minimizing the computational cost in solving the equations of motion and integrating is equivalent to finding a minimum weight spanning tree. The SCBST sub-problem is defined and related to the MRST problem. An algorithm of polynomial complexity has been shown to solve both problems.

The MCCST problem and some related problems (SCBMWST, MWSCBST, BWSCBST and BCBMWST) are shown to be NP-hard by proving that the decision problem, BCBBWST to be NP-complete. Heuristic algorithms for the MCCST problem were developed. Implementation of the proposed algorithms and simulation results for varying instances of the problem were presented.

It has been shown that the heuristic algorithms presented may fall into some local minimum. It was also shown that the proper selection of heuristics may reduce the chance of this difficulty. However, there are no criteria for selecting adequate heuristics. The process of improving the heuristics is shown by comparing two heuristics.

Heuristics which start from an MWST of a graph were not discussed in this paper. In general, this approach will yield a better initial spanning tree (i.e. the spanning tree has

smaller cost funstion), but may have a slower rate of convergence. A comparative study of these heuristic algorithms for solving MCCST problems is the subject of the authors current endeavors.

References

Bae, D.S., and Haug, E.J., "A Recursive Formulation for Constrained Mechanical Systems, Part I - Open Loop," *Mechanics of Structures and Machines*, Vol. 15 (4), 1987.

Bae, D.S., Hwang, R.S., and Haug, E.J., "A Recursive Formulation for Real-Time Dynamic Simulation," Technical Report D-2, Center for Computer Aided Design, The University of Iowa, Jan, 1988.

Booth, H., and Westbrooke, J., 1994, "Linear Algorithm for Analysis of Minimum Spanning and Shortest-Path Trees of Planar Graphs," *Algorithmica*, Vol. 11, No. 4, pp. 341-352.

Christofedes, N., "Graph Theory: An Algorithmic Approach," Academic Press Inc., 1986.

Easwarakumar, K.S., Pandu Ranga, C., and Cheston, *G*.A., 1994, "Linear Algorithm for Centering a Spanning Tree of a Biconnected Graph," *Inf. Process. Letters*, Vol. 51, No. 3, pp. 121-124.

Garey, M. R. and Johnson, D. S., *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, SF, CA, 1979

Haug, E. J., 1995, Private Communication, University of Iowa, Iowa City, IA, 52242.

Ho, J.M., Lee, D.*T*., Chang, C.H., and Wong, C.K., 1991, "Minimum Diameter Spanning Trees and Related Problems," *SIAM J. Comput*., Vol. 20, No. 5.

Hwang, R.S., and Haug, E.J., 1989, "Topology Analysis of Multibody Systems for Recursive Dynamics Formulation," *Mech. Struct. and Mach.*, (Haug, E.J., ed.), Vol. 17, No. 2, pp. 239-258.

Iwano, K., and Katoh, N., 1993, "Efficient Algorithms for Finding the Most Vital Edge of a Minimum Spanning Tree," *Information Processing Letters*, Vol. 48, No. 5, pp. 211-213.

Johnson, D.B., and metaxas, P., 1992, "Parallel Algorithm for Computing Minimum Spanning Trees," *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA.

Khuller, S., and Schieber, B., 1992, "On Independent Spanning Trees," *Information Processing Letters*, Vol. 42, No. 6, pp. 321-323.

Khuller, S., Raghavachari, B., and Young, N., 1993, "Balancing Minimum Spanning and Shortest Path Trees," *Proceedings of the Fourth Annual ACM-SIAm Symposium on Discrete Algorithms*, New York, NY.

Kim, S.S., "A State Space Formulation for Multibody Dynamic Systems Subject to Control," Ph.D. Thesis, The University of Iowa, 1984.

NADS, 1995, National Advanced Driving Simulator Reference Manual, Center for Computer Aided Design, The University of Iowa, Iowa City, IA.

Pape, U., 1980, "Program for Finding the Shortest Path from a Specific Node to All other Nodes in a Network," *ACM TOMS*, Vol. 6, No. 562, pp. 450-455.

Park, *T.G.*, and Oldfield, J.V., 1993, "Minimum Spanning Tree Generation with Content-Addressable Memory," *Electronics Letters*, Vol. 29, No. 11, pp. 1037-1039.

Pawagi, S., and Kaser, O., 1993, "Optimal Parallel Algorithms for Multiple Updates of Minimum Spanning Trees," *Algorithmica*, Vol. 9, No. 4, pp. 357-381.

Roberson, R.E., "The Path Matrix of A Graph, Its Construction and Its Use in Evaluating Certain Products," *Computer Methods in Applied Mechanics and Engineering*, Vol. 42, 1984, pp. 47-56.

Saitoh, A., Tsujino, Y., and Tokura, N., 1992, "Complete Spanning Tree Maintenance Algorithm and its Complexity," *Systems and Computers in Japan*, Vol. 23, No. 11, pp. 319-328.

Sheth, P.N., and Uicker, J.J., "IMP (Integrated Mechanical Programs): A Computer-Aided Design System for Mechanisms and Linkages," J. Engr. Industry, Vol. 94, No. 2, 1972, pp. 454-464.

Suraweera, F., 1989, "A Fast Algorithm for the Minimum Spanning Tree," *Computers in Industry*, Vol. 13.

Suraweera, F., Bhattacharya, P., 1993, "$O(\log m)$ Parallel Algorithm for the Minimum Spanning Tree Problem," *Information Processing Letters*, Vol. 45, No. 3, pp. 159-163.

Tsai, F. F. and Haug, E. J., "Automated method for high speed simulation of multibody dynamic systems," *Tech. report R47*, Center for Simulation and Design Optimization of Mech. Systems, The University of Iowa, Iowa City, 1989.

Tsin, Y.H., 1993, "Incremental Distributed Asynchronous Algorithm for Minimum Spanning Trees," *Computer Networks and ISDN Systems*, Vol. 26, No. 2, pp. 227-232.

Wittenberg, J. and Wolz, U., "MESA VERDE: A Symbolic Program for Nonlinear Articulated-Rigid-Body Dynamics," ASME, 85-DET-151.

Wittenberg, J., "Dynamics of Systems of Rigid Bodies," B. *G*. Terbner, Stuttgart, 1977.

Wittenberg, J., "Nonlinear Equations of Motion for Arbitrary Systems on Interconnected Rigid Bodies," *Symposium on the Dynamics of Multibody Systems*, Munich, Germany, Pro. published by Spring-Verlag, 1978, (K. Magnus, ed.)

Yao, A.C., 1975, "An $O(E \log \log V)$ Algorithm for Finding Minimum Spanning Trees," *Inf. Process.*, Vol. 4.

Zeid, A. and Overholt, J. L., 1995, "Singularity Pperturbed Bond Graph Models for Simulation of Multibody Systems," ASME *Journal of Dynamic Systems, Meausurements, and Control*, Vol. 117,, No. 3, pp. 401-410.

Zeid, A., 1989, "Bond Graph Modeling of Planar Mechanisms with Realistic Joint Effects," ASME *Journal of Dynamic Systems, Meausurements, and Control*, Vol. 111, pp. 15-23.