An Exact Parallel Algorithm for Scheduling When Production Costs Depend on Consecutive System States

by

J.F. Pekny, D.L. Miller, G.J. McRae

EDRC 06-52-89 3

An Exact Parallel Algorithm for Scheduling When Production Costs Depend on Consecutive System States

J. F. Pekny[t]

D. L. Miller[%]

G. J. McRae[t]

ABSTRACT

An exact parallel algorithm is presented for determining schedules when costs depend on consecutive jobs passed through the production system and the benefits of production are the sum of the benefits associated with each job. The algorithm provides optimal schedules when the goal of production is to (i) maximize benefits minus costs, (ii) minmize costs (iii) maximize sum of benefits while keeping costs below a prescribed level, or (iv) minimize costs while attaining some level of benefits. The scheduling algorithm is shown to be equivalent to the prize collecting traveling salesman problem. The algorithm uses a branch and bound approach based on a Lagrangian lower bounding technique, branching rules that partition the search tree, and an upper bounding procedure based on a patching algorithm. The algorithm is implemented on a shared memory multiprocessor and computational results are presented for problems ranging in size from 50 to 200.

Key Words: Scheduling, Branch and Bound, Parallel Algorithm, Generalized Traveling Salesman Problem

November 7, 1988

t Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh PA, 15213

% Central Researdi and Devlopment Department, E. I. du Pont de Nemours & Co., Inc., Wilmington DE, 19803

An Exact Parallel Algorithm for Scheduling When Production Costs Depend on Consecutive System States

/. F. Peknyt

D. L. MiUert

G. *J. McRaet*

Introduction

A number of scheduling situations may be viewed as a problem of choosing which jobs should be processed in a fixed time frame and sequencing those jobs which are to be processed. Associated with a particular sequence of production is some benefit and cost. Depending on how the benefits and costs are measured there are several rationale objectives for choosing a particular sequence of production:

(1) maximize the sum of benefits minus the sum of costs.

(2) minimize production costs without regard for benefits.

(3) maximize benefits such that costs do not exceed some fixed amount.

(4) minimize costs while attaining some level of benefits.

Each of these four objectives is justifiable in some production environment. If both benefits and costs are measured in common units such as dollars, then the first objective seems most appropriate. However, if production costs are measured in units of some finite resource such as available machine or labor time, then the second or third objective may be the most appropriate. When the benefits of production activities are indirect, then the fourth objective of attaining some target level of benefit may be the objective of choice. This paper shows that with certain restrictions on the nature of benefits and costs associated with job processing, all four objectives are treatable using the same basic sequencing algorithm.

In the scheduling scenario addressed by this paper there are a set of N jobs labeled (1.....N). These jobs are candidates for processing through a system where costs depend only on the consecutive jobs in the production sequence. A flowshop with the zero wait processing condition (see **[1]),** and

t **Department** of Chemical Engineering. Carnegie Mellon University, Pittsburgh PA, 15213

X **Central Research and** Devlopment Department, E. I. du Pont de Nemours & Co., Inc., Wilmington DE, 19803

single machine scheduling with setup costs (see [2]) are two examples of such a system. Thus if job j is processed after job i, a production cost c_{xj} is incurred. Matrix c = (c_{tJ}) for all ij € (1,...,N) is defined to be the transistion cost matrix since it contains the costs due to system transistion between all possible pairs of jobs. The diagonal element c_u of the transistion cost matrix represents the cost incurred if a job i is not processed by the system during the time horizon of the model. Every job i which is processed by the scheduling system contributes benefit *p%*. Startup and shutdown costs are incorporated into the scheduling model by introducing an artificial job 0. Cost c_Oj represents the cost of starting production with job j 6 (1,...,N) and cost c_{i0} represents the cost of shutting down production after job i 6 {1,...,N}. The transistion cost matrix is augmented so that row zero contains costs *C&* for i € {1,...,N} and column zero contains costs $c_{/0}$ for j *e* {1,...,N}. Transistion cost matrix element *CQQ* is set to infinity to indicate that at least one job will be processed. Also, there is no benefit associated with artificial job 0 so that p_0 is zero. The scheduling scenario is completely defined by n, the number of jobs to be processed, the augmented transistion cost matrix c, and the job benefit vector p. A solution to the scheduling scenario consists of a list of which jobs are not processed during the time horizon of the model and a sequence of production for those jobs which are processed.

An optimal solution to the scenario described in the previous paragraph depends on which of the four scheduling objectives is used. To illustrate the impact of the scheduling objective, consider the example shown in Figure 1. In this example there are nine jobs labeled (1,...,9) in addition to artificial job 0. As Figure 2 shows the optimal production sequence depends on which of the four scheduling objectives is appropriate. The following sections will show how the same algorithm may be used to determine optimal sequences for all four scheduling objectives.

Problem Formulations

The scheduling scenario described in the introduction may be represented by a complete directed graph G= (V,A), where vertex set V = (0,l,...,N) represents the set of jobs including artificial job 0 and arc set A= {(i,j) I i,j € V} represents precedence in production. In terms of graph G, a feasible schedule is a subgraph G'= $(V\backslash A^f)$ where V^f = V, the cardinality of A' is N+ 1, each vertex has an indegree and outdegree of one, and arc set A' defines a feasible production sequence. If arc (i,j) € A' with i * j and i,j e V, then job i precedes job j in a production sequence. If arc (i,i) e A' for i *e* {1.....N} then job i is not processed within the time horizon of the model. Arc (Oj), j€ (1.....N) indicates that job j is the first job to be processed and arc (k,0), k € {1,...,N} indicates that job k is the last job to be processed. Figure 3 shows a subgraph representing a feasible schedule and a subgraph representing an infeasible schedule. As Figure 3 shows, a feasible schedule is represented by a graph consisting of a set of loops (possibly empty) and a single cycle involving two or more vertices of the subgraph. The elements of the transistion cost matrix have a one to one correspondence with arc set A. Thus *Cij* is the cost incurred if arc (ij) is present in the subgraph representing a feasible solution.

Likewise, there is a one to one correspondence between the benefit vector and the vertex set V. In terms of subgraph G' benefit A is realized if vertex i does not participate in a loop. By associating binary integer variable x_{ih} ij *e* V with the arcs of set A, the scheduling scenario presented in the introduction may be formulated as an integer program. In the integer programming formulation, x_{cJ} is one if arc (i,j) is present in the optimal solution and zero otherwise. The goal of maximizing the sum of benefits minus the sum of costs may be expressed as:

$$\max_{i*V} 2>a-*«)-\sum_{imVjmV} Z^{c}\wedge/ \qquad (1)$$

The left term in equation (1) represents the sum of benefits and the right term represents the sum of costs. Equation (1) may be written more compactly as:

$$\sum p_i - \min \sum X \wedge \qquad (2)$$

where

lij=Cij for *each ije V : i* j*

Zu^Cii+Pi for each ieV

Since the first summation in equation (2) is a constant, it may be dropped to yield the following objective that will maximize the sum of benefits minus the sum of costs:

$$\min \sum_{i\in V} \sum_{j\in V} \tilde{c}_{ij} x_{ij} \qquad (3)$$

Equation (3) is subject to the following constraints which specify the scheduling scenario of the introduction:

$$lx_{ir}h \qquad j*V \qquad (4)$$

$$\sum_{j\in V} \wedge = 1, \quad ieV \qquad (5)$$

$$\sum_{ic5/*«5\backslash\{i\}} Z \ *«/£|Shl. \quad \textit{for each } SczV \qquad (6)$$

XijG (0,1}, for *each i Je V* (7)

The following objective in combination with equations (4-7) will minimize production costs without regard for the production benefits:

$$\min \sum_{i\in V} \sum_{j\in V} c_{ij} x_{ij} \qquad (8)$$

Note that equations (3-7) and equations (4-7,8) are almost identical to the well known Asymmetric Traveling Salesman Problem (ATSP). The review presented in [3] is an excellent summary of ATSP

research up to 198S. The parallel algorithm reported in [4] has been used to solve large randomly **generated ATSPs. The objective** of maximizing benefits such that costs do not exceed some fixed amount Cmu **may be stated as:**

$$\min \sum p_i x_{ii} \tag{9}$$

This objective is subject to equations (4-7) as well as the following equation:

$$\sum_{i \in V j \in V \backslash} X \; {}^c n^x ij \; * \; {}^c m « \tag{10}$$

Equations (4-7,9,10) are equivalent to the Orienteering Problem reported in [5,6,7]. The fourth objective listed in the introduction of minimizing costs while attaining some level of benefits also uses equation (8) as the objective function. The following equation is necessary to enforce the goal of attaining a minimum level of benefits, P_{mia}:

$$ZtPiXu* \; EA\text{-}\hat{}\min \tag{11}$$

Equations (4-8,11) is an integer programming representation of the Prize Collecting Traveling Salesman Problem (PCTSP) reported in [8,7]. When phrased in PCTSP terminology, p_4 of equation (11) is the prize received by the salesman for visiting city i. P_{mm} is the minimum sum of prizes that the salesman must collect in his travels. The salesman's goal is to order the cities he will visit in such a way as to minimize total travel costs. The salesman incurs cost c_{tj} for traveling between city i and city j. Cost $c_{u'}$ is incurred if the salesman does not visit city i.

Table 1 summarizes the four scheduling objectives along with the corresponding equation numbers comprising an integer programming formulation. Table 1 also associates the names of the combinatorial problems as they are known in the operations research literature with the corresponding scheduling objective. All four of the integer programming problems listed in Table 1 can be solved by an algorithm for the PCTSP (integer programming problem 4). By setting P_{mixi} equal to zero, the PCTSP formulation becomes equivalent to integer programming problems 1 and 2. Integer programming problem 3 can be solved using an algorithm for the PCTSP parametrically. Parametric application of the PCTSP algorithm is detailed below. The next section describes an exact parallel algorithm for the PCTSP. Since the PCTSP is an NP-hard problem (see [7]), the algorithm uses a branch and bound approach. A parallel algorithm is given for the PCTSP since the problem is computationally demanding.

Examination of the four integer programming formulations summarized in Table 1 suggests the Resource Constrained Traveling Salesman Problem (RCTSP). In the terminology of the traveling salesman problem (see [3]), the RCTSP consists of finding a tour of minimum cost that docs not necessarily visit all cities. The resource usage of this tour cannot be more than R_{mM}. Traveling between city i and city j requires a resource expenditure of $r_{i\}}$ and costs c_{ir}. As an integer program, the RCTSP may be represented as follows using the complete graph G= (V,A) defined above:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{12}$$

$$\sum_{imV} x_{iy} = 1, \quad jeV \tag{13}$$

$$\sum_{j*V} x_{ij} = 1, \quad i \in V \tag{14}$$

12>i>*iy£*m« (15)
iaVjaV

Z Z *v*FM. for *each SczV* (16)
*i*Sj*S\{i)*

Xije[0M for *each iJeV* (17)

Interpreted in scheduling terminology, the cities of the RCTSP correspond to jobs. City 0 corresponds to artificial job 0 which indicates the beginning and end of the production sequence. The costs $c_{l\}}$ correspond exactly to the augmented transistion cost matrix. The resource usage $r_{f\}}$ can be associated with the amount of resource required to switch the production facility from job i to job j. The objective of the RCTSP when viewed as a scheduling problem is to minimize production costs such that not more than a set amount of some resource is consumed. An RCTSP cannot be directly solved as a PCTSP as the other integer programs of Table 1. However, PCTSP can be solved as an RCTSP by setting $r_{\overline{u}}$-p_i for all ieV, fy=0 for all ij e V : i* j, and R_{mtx} = £A—^min* Thus each of the problems listed in
ieV
Table 1 is a special case of RCTSP.

An Exact Parallel PCTSP Algorithm

The parallel PCTSP branch and bound algorithm is patterned after the processor shop model reported in [4]. Figure 4 shows the processor shop dataflow. Figure 5 shows the control algorithm each processor uses when operating in the dataflow environment of Figure 4. Each of the components shown in Figure 4 are explained in the sections that follow. Complete details and justification for using the processor shop model as the basis for parallel branch and bound may be found in [4]. A key component of the model shown in Figure 4 is the Lagrangian assignment problem data type (LAP). This data type encapsulates information necessary for each of the algorithm components to function. Figure 6 provides an outline of the information present in an LAP. A processor contributes to PCTSP solution by performing operations indicated by the algorithm components on various LAPs. The LAPs correspond to the nodes of the branch and bound search tree. LAPs are characterized as being solved or unsolved. The lower bounding procedure has been applied to solved LAPs, but not to unsolved

LAPs. **The two queues** shown in Figure 4 are accessed using a best bound first strategy. Thus a processor **removes the LAP** with the least lower bound first. LAPs resident in the unsolved queue inherit their parent's lower bound for purposes of determining order of removal.

Lower Bounding Technique

Lower bounds for the PCTSP algorithm are calculated using the Lagrangian relaxation method [9]. The Lagrangian problem is formed by neglecting equation (6) and dualizing equation (11) in the PCTSP formulation of equations (4-8,11). The resulting Lagrangian problem may be represented as follows:

$$\max_{X*0} L(\lambda) \qquad (18)$$

where

$$L(\lambda) = -XU + \min \text{££}_{i\in V j\in V} \hat{c}(\lambda)_{ij} x_{ij} \qquad (19)$$

$$U = \sum_{i*V} p_i - P_{\min} \qquad (20)$$

$\hat{c}(\lambda)_{ij}$ = *Cij* for *each iJeV* : ż* ; (21)

$\hat{c}(X)_u$ = *Cu+Xpi* for *each ieV* (22)

subject to

Ex,;=l, ***jeV*** (23)

£xtf=l, ***ieV*** (24)

Xije {0,1}, for *each ije V* (25)

For fixed X, the integer program represented by (18-25) is simply an assignment problem which is solved using any of the algorithms reported in [10,11,12]. As is evident by equation (18), the Lagrangian problem is simply a single variable nonlinear maximization problem. An algorithm for obtaining a good solution to this maximization problem consists of a bounding phase and a bisection phase. During the bounding phase X_L and $\backslash_v$ are determined such that :

(i) the solution (x_L) corresponding to $L(\backslash_L)$ does not satisfy equation (11).

(ii) the solution *(xy)* corresponding to L(X(/) does satisfy equation (11).

The bounding **phase** requires **an** initial value of X which is increased by *STEPJSIZE* if the corresponding solution **(x) does** not satisfy equation (11). If (x) satisfies equation (11) then *X* is decreased by *STEPJSIZE.* As indicated by equation (18), X must be non-negative. Thus *X* is set to zero if a decrease in X would ever cause X to be negative. The bounding phase continues increasing or decreasing X in this fashion until an appropriate X_L and X_v are found that satisfy (i) and (ii). The X associated with the lower bound of the parent node in the branch and bound tree is used as an initial value for X. If the node being lower bounded does not have a parent, then an initial X is computed via the technique suggested in [7]. *STEPJSIZE* is initialized to be *lambda/2.0* or 1.0 if the initial lambda is 0. After each increase or decrease of X, *STEPJSIZE* is set to *2*STEP_SIZE* in order to accelerate the search for a X_L and X^. During the bisection phase X_L and X_v are updated such that (i) and (ii) remain satisfied. This update is performed via a bisection technique where L(X_M) and the corresponding *($^X M$)* are determined with X^ = *(X_L* + X</)/2. If *{x_M)* satisfies equation (11) then X_M replaces *Xy* otherwise X_L is replaced. The bisection technique terminates normally when X</ - X_L is less than *t*Xu.* For the computational results reported below, an e of 0.1 has been found to lead to good performance. The Lagrangian method is terminated immediately if, at any time, the value of the objective function (18) exceeds the global upper bound or if the solution (x) corresponding to X = 0 satisfies equation (11). The Lagrangian method is also terminated if the number of evaluations of objective function (18) exceed MAXJTERATIONS with MAXJTERATIONS set at 20. When the Lagrangian method terminates, the largest value of the objective function (18) is reported to be the lower bound.

In the course of computing a near optimum to equation (18), several intermediate values for X must be computed. Every new value of X does not require a complete assignment problem solution. Successive assignment problems corresponding to new values of X can be solved parametricaily based on the most recent assignment problem solution. By solving assignment problems parametricaily in this fashion, the computational efficiency of the lower bounding technique is greatly improved.

Branching Rules

In terms of the dataflow diagram shown in Figure 4, the branching rules take a solved LAP and create one or more unsolved LAPs which are placed in the unsolved queue. The branching rules operate on a near optimal solution x to equations (18-23) whose description is contained in a solved LAP. Solution x does not necessarily satisfy equations (6,11). Figure 7 illustrates a graph corresponding to an x which violates equations (6,11). The solution shown in the graph of Figure 7 violates equation (6) because there are multiple subtours. The solution shown in Figure 7 violates equation (11) because the £A*ii$_{imV}$ exceeds $\sum_{i \in V} p_i - P_{min}$.

The branching rules **are** enforced using include and exclude arcs which refer to graph G= (V,A) **defined** in **the formulation** section. The binary variable $x_{\%J}$ associated with an include arc (i,j) is forced to be one in **the** solution of the lower bounding equations (18-25). The binary variable x_{1f} associated with an exclude **arc** (ij) is forced to be zero in the solution of the lower bounding equations (18-25). As Figure 6 shows include and exclude arc sets are stored in LAPs. [13,4] explain how include and exclude arcs are enforced in assignment problem solutions that yield the lower bounds.

There **are** three types of branching rules used by the PCTSP algorithm:

(1) loop reduction - eliminate loops for all i e V such that p_x > £A-^min and x_u = 1.
i∊V

(2) loop branching - used when £AXU > $\sum_{i \in V} p_i - P_{min}$.
ieV

(3) subtour branching - used when £ £ $^{x}ij^{>}$ 1$ h 1 f°^{r} some S **c** V.
ieS/€S\{i)

Loop reduction and loop branching seek to enforce equation (11) while subtour branching seeks to enforce equation (6). Note that loop reduction is a special case of loop branching. The branching rules are precisely stated using solution x*, include arc set /*, and exclude arc set E_k from solved LAP_k. The include arc set I_k implies a set F_k of vertices that are forced to participate in loops. Given these definitions, the branching rules are as follows:

loop reduction - Let L= (ti,*2»*.-,'$_{r}$} be the largest vertex set such that for all ieL: (a) p_g >

2 *Pi~Pmin>* (b) $x^k u$ = U and (c) arc (i,i) is not a member of I_k for all i€L. Equation (11) implies
i*V\Fk

x^^Oandx^^Oand ••• andx^^O (26)

Generate one successor LAP_{kx} of LAP_k defined by

$$E_{k1}=E_k \cup \{(i_1,i_1),(i_2,i_2),\ldots\text{»0'M*I)}\}$$
$$I_{k1}=I_k$$

loop branching - Let L= (ii,'2t—t^{f}'r) $^{be\ the}$ largest vertex set such that: (a) £AX*$_{U}$ > £ A- P_{min}.
ieL ***i€V\Fk***

(b) $x^k u$ = 1 for all ieL, (c) arc (i,i) is not a member of I_k for all ieL. Equation (11) implies

**^{k}i$_{x}$irŌ or x'yfO or • • • or x^{k} ^=0* (28)

Generate t successors LAP_{kXy}...,LAP_{kt} of LAP_k defined by

E_{kr}=E_{kKJ} {(wV)} 1
hr-hKJ Kil.il). . . . (ir.l.ir-l)}J^{fOrrSl}--^{r} (29)

Note that for r=l, the include arc set I_{kr} is simply equal to /*. Computational experience shows that the most efficient implementation of this branching rule labels the vertices of L in order of descending prize **value. Thus the labels** {iW—^} of set L are always assigned such that $p_1 \geq p$^ • • · $\geq p_{ir}$

subtour braching (see [14,15,16,17,13]) - Let A_s = **{(i$_t$»i2).....(4.«i)} be** the arc set of a subtour of x^h containing a minimum number of arcs not in I_k involving vertex set S=* {ii,...,/,}. Equation (6) implies

$$\sum_{(i,j)\in A_s} x_{ij} \leq |S|-1 \qquad \textbf{(30)}$$

Without loss of generality, assume A_s - /* = {(I'I, 1*2)• ••>(**.i*+i))» ^{w}^h s £ t. Then equation (30) implies

(**,y$_2$=0) *or* (x*$_{<l12}$=l,x*$_{12<3}$=0) or • • *or* (x^k^ • • • ^{s}x_$_A$=l, x*$_{w+1}$=0) (31)

where s+ 1 is to be taken modulo t. Generate s successors, $LAP_k i,..._t LAP_{kt}$ of LAP_k defined by

$$\left.\begin{array}{l} E_{kr}=E_k \cup \{(i_r,i_{r+1})\} \\ I_{kr}=I_k \cup \{(i_1,i_2),\cdots,(i_{r-1},i_r)\} \end{array}\right\} \text{for } r=1 \ldots\ldots J \qquad (32)$$

The branching rule that creates the least number of unsolved LAP is applied whenever two or more of the branching rules are applicable. By this criteria, loop reduction is always preferentially applied since it creates only one unsolved LAP. Note that each of the three branching rules partitions the branch and bound tree in the sense that no two nodes will be redundant.

Elimination Rules

The elimination rules are used to fathom branches in the branch and bound tree. As Figure 4 illustrates, the elimination rules are applied to an LAP before any amount of substantial computation is performed. This guarantees that a minimum amount of computational effort is expended in the search for an optimal solution. There are two elimination rules. The first rule eliminates an LAP from consideration whenever its lower bound equals or exceeds the current global upper bound. The second rule eliminates an LAP from consideration whenever the LAP's include arc set implies that equation (11) cannot be satisfied. This occurs whenever $\underset{ieL}{\text{£}}$p,-> $\underset{ieV}{\text{^A}}$-^min where L= {i$_{lf}$ i$_{2t}$..., i_t] is the **set** of vertices that are involved in loops that are members of an LAPs include arc set.

Upper Bounding Technique

The **upper** bounding technique provides approximate solutions to equations (4-8,11). In combination with the elimination rules, the upper bounding technique permits fathoming of search tree branches that do not lead to an optimum. A discussion of the role of the upper bounding technique in a parallel branch and bound algorithm is given in [4]. The upper bounding technique used in the PCTSP algorithm is an extension of the patching algorithm given in [18]. As discussed above, the solution *x* to the lower bounding equations (18-25) can be interpreted on graph G as a collection of loops and subtours. The PCTSP patching algorithm operates on solution x in two phases:

(i) The patching algorithm reported in [18] combines the subtours of solution *x* to form a single tour. The loops are not considered in this phase. Thus solution *x* has been modified to consist of a single tour and a collection of loops. Denote this modified solution i. If £ satisfies equation (11), phase (ii) does not execute since a valid solution has been found to equations (4-8,11).

(ii) This phase satisfies equation (11) by incorporating some of the vertices that participate in loops into the single tour formed in phase (i). Let L= {/i/2>—»*r) be the vertices that participate in loops. Let /,· be such that £,/.= 1 for all i€ V. Variable /,· is known as the successor of i since arc (i«/i) is present in the solution subgraph of graph G (see the formulation section above). The cost of incorporating a loop vertex *ieL* into the single tour from phase (i) is expressed as:

Cif+tji-Cu-Cjf. **(32)**

where *jeV\L* is some vertex that is a member of the tour. If the quantity expressed in relation (32) is negative or zero for any *IeL* and *jeV\L* then vertex i is incorporated into the tour by setting £,/= 1, iy;= 1, £u=0, and i>/.= 0. Additionally, /; is set to *fj* and then *f,* ·is set to i. Figure 8 illustrates the process of incorporating a loop into a tour. If expression (32) is not zero or negative for any *ieL, jeV\L* then an *IeL,]eV\L* is found that maximizes the following relation:

PiUcif+Cji-Cu-Ctfj) **(33)**

The loop vertex represented by *I* is incorporated into the tour using / as shown in Figure 8 and by setting Jfy^l, J?£=l, *£=0, *//:=0, /;=//, and then /;=?. The steps of phase (ii) are repeated until equation (11) of the PCTSP formulation is satisfied.

The solution x corresponding to the global upper bound is replaced by the PCTSP solution $\hat{x}$ produced by the patching procedure if £ Z c«>/V· is less than £ Zc»>x»>- The patching procedure described by (i) and (ii) has a worst case computational complexity of \V |^3. In practice, the patching procedure need not be applied to every LAP. Once a good upper bound is established, repeated application of the patching procedure is a waste of computational resources. A procedure similar to that described in [4]

can be used to selectively apply the patching procedure.

PCTSP Algorithm Computational Results

The parallel PCTSP algorithm described above was implemented on a 14 processor BBN Butterfly Plus computer! possessing 56 megabytes of shared memory. The Butterfly Plus is a tightly coupled shared memory multiprocessor comprised of Motorola 68020/68881 processors accessing 4 megabytes of local memory and nonlocal memory through a packet switched network. Remote memory accesses are transparent to processors, but slower than local memory accesses. The Butterfly Plus computer does not support simultaneous access to individual memory locations. When two or more requests are made for reading **a** memory location only one access is serviced. The other requests must be retried at a later time. The algorithm was implemented in the C programming language under a Chrysalis operating system environment. A more complete description of the Butterfly Plus computer may be found in [19].

The PCTSP algorithm was tested on problems with cost matrix elements drawn from a uniform distribution of integers in the range [0,1000]. The elements of the prize vector p in equation (11) were drawn from a uniform distribution of integers in the range [0,100]. The target prize P_{min} was set to be <**ZA- Problems were generated for an **a** of 0.2, 0.5, 0.8, 1.0 and n= \V\ of 50, 100, 150, and **200.**
imV
The PCTSP algorithm was used to solve 10 problems for each n and a.

Table 2 contains the average and standard deviation of the total execution time, number of nodes evaluated in the branch and bound tree, and the ratio of the optimal solution value to the lower bound value of the root node for each **a** and n. A plot of the total execution time as a function of n for each **a** is given in Figure 9. The ratio column of Table 2 shows that the strength of the Lagrangian lower bounding technique increases with both increasing a and n. [7] discusses ways in which the lower bounds can be improved for both small n and a. Figure 9 and Table 2 suggest that the most difficult problems possess an **a** between 0.5 and 0.8 as evidenced by both total execution time and number of nodes evaluated in the branch and bound tree. The case of a= 1.00 shows that the pure ATSP is significantly easier to solve than the PCTSP. In fact for a= 1.00 the lower bounding technique described above can be replaced by an assignment problem based technique. [4] describes a parallel algorithm for the ATSP based on an assignment problem bounding approach. Note that both the total execution time and number of nodes evaluated sometimes demonstrate considerable variability among problems of the same **a** and n.

t **Butterfly Plui is a trademark** of Bolt, Beranek, and Newman Inc.

In **order to gauge the** benefit of parallelism, the same 10 problems were solved for an n of 100 and **an a** of 0.2, 0.5, 0.8, and 1.0 using different numbers of processors. Table 3 lists the results as a function **of the number** of processors and a. The data of Table 3 is plotted in Figure 10. In addition to **reducing average** execution time, algorithm parallelism also reduces execution time variability. Algorithm parallelism has exactly the opposite effect on the number of nodes evaluated in the branch and bound **tree.** As Table 3 shows, both the average and standard deviation of the number of nodes evaluated increases with an increasing number of processors. An increased number of nodes evaluated tends to offset the benefit of adding additional processors. As Table 3 and Figure 10 indicate, algorithm speedup ranges from about 2.89 for a= 1.0 to about 9.1S for as 0.8 using 14 processors. The storage scheme for cost matrix c provides an additional explanation for why the parallel algorithm does not take full advantage of additional processors. The cost matrix is distributed across all processors participating in problem solution. Each processor maintains roughly an equal portion of the cost matrix in its local memory. In the course of problem solution each processor may address the entire cost matrix. Thus multiprocessor executions of the algorithm suffer a performance penalty due to nonlocal cost matrix access. This problem may be circumvented by storing a complete copy local to each processor. However maintaining multiple copies of the cost matrix becomes prohibitive as problem size grows.

Parametric Algorithm for the Orienteering Problem

As pointed out above, the Orienteering Problem represented by equations (4-7,9,10) may be solved by applying the PCTSP algorithm parametrically. In this parametric solution, *Pmin* of equation (11) is modified in a fashion analogous to a binary search until an optimal Orienteering solution is found. The following algorithm states precisely how the PCTSP algorithm is used to solve the Orienteering problem:

```
Algorithm SolveOrienteer

Input:
problem size n, cost matrix c, benefit vector p, maximum tour cost $C_{max}$.

Output:
A tour that collects a maximum amount of benefits that doesn't incur a cost greater than $C_{max}$.

begin
  best_benefits := -<»;
  lower_bound   := 0;
  upper .bound := $\sum_{i \in V} p_i$;
```

```
    **while(k>werj>ound £ upper.bound)**
      **begin**
        mid_bound := (lower J>ound + upper_bound)/2;
        set ^min #:= midj>ound;
        let x be the PCTSP solution to equations (4-8,11) using $n,c,p,P_{min}$;


          upper_bound := mid_bound - 1;
        **else**
          **begin**
            benefits := $\sum_{i \in V} p_i(1-x_{ii})$;
            if (benefits > best_benefits) let best_benefits:= benefits, save x;
            lower_bound := $\sum p_i(1-x_{ii})$;
          end
      **end**
    if (bestj)rize = -«)
      orienteering problem has no solution;
    **else**
      return best_prize and the associated x;
  **end**
```

The efficiency of Algorithm SolveOrienteer can be improved upon if the PCTSP algorithm is modified to include the constraint represented by equation (10). The modification takes the form of an additional elimination rule. Thus if any lower bound calculated during PCTSP solution exceeds C_{mtx}, the corresponding LAP is eliminated from further consideration. When the modified PCTSP algorithm is not used to solve Orienteering Problems, C_{mtx} can be set to infinity. Note that the PCTSP can be solved parametrically using an exact Orienteering algorithm in a manner entirely analogous to that illustrated in Algorithm SolveOrienteer.

Conclusions

A parallel algorithm has been presented for the PCTSP. This algorithm has application to scheduling scenarios where production costs depend only on consecutive jobs passed through the system and overall benefits of production are the sum of the benefits for each job that is processed. The PCTSP model offers flexibility in terms of scheduling to meet various objectives provided the assumptions on production costs and benefits are met. Computational results show the parallel PCTSP algorithm to be effective on randomly generated problems ranging in size from SO to 200 cities. The RCTSP generalization of the PCTSP formulated above promises to be an even more powerful tool for providing optimal solutions to scheduling scenarios. Solution procedures for the RCTSP can be based on an approach similar to that for the PCTSP.

References

1. J. N. D. Gupta, "Optimal Flowshop Schedules with No Intermediate Storage Space/' *Naval Research Logistics Quarterly,* vol. 23, pp. 235-243, 1976.
2. K. R. Baker, *Introduction to Sequencing and Scheduling,* John Wiley, New York, 1974.
3. E. L. Lawler, J. L. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization,* John Wiley & Sons, New York, 1985.
4. J. F. Pekny and D. L. Miller, "A Parallel Branch and Bound Algorithm For Solving Large Asymmetric Traveling Salesman Problems," *Mathematical Programming,* 1988. submitted
5. T. Tsiligirides, "Heuristic Methods Applied to Orienteering," 7. *Opl Res. Soc,* vol. 35, pp. 797-809, 1984.
6. Bruce L. Golden, Larry Levy, and Rakesh Vohra, "The Orienteering Problem," *Naval Research Logistics,* vol. 34, pp. 307-318, 1987.
7. Matteo Fischetti and Paolo Toth, "An Additive Approach for the Optimal Solution of the Prize-Collecting Travelling Salesman Problem," *Research Report OR/87/4,* D.E.I.S, University of Bologna, Italy, May 1987.
8. Egon Balas, "The Prize Collecting Traveling Salesman Problem," *Management Science Research Report No. MSRR-539,* Carnegie Mellon University, Pittsburgh PA, July 1987.
9. Marshall L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science,* vol. 27, pp. 1-18, January, 1981.
10. G. Carpaneto and P. Toth, "Primal-Dual Algorithms for the Assignment Problem,' *Discrete Applied Mathematics,* vol. 18, pp. 137-153, 1987.
11. S. Martello and P. Toth, "Linear Assignment Problems," *Annals of Discrete Mathematics,* vol. 31, pp. 259-282, 1987.
12. E. L. Lawler, *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart, and Winston, New York, 1976.
13. G. Carpaneto and P. Toth, "Some New Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem," *Management Science,* vol. 26, pp. 736-743, 1980.
14. E. Balas and P. Toth, "Branch and Bound Methods," in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization,* ed. E. L. Lawler et al., John Wiley & Sons, New York, 1985.

15. M. Bellmore and M. J. C. Malone, "Pathology of Traveling Salesman Subtour-Elimination Algorithms/[9] *Operations Research*, vol. 19, pp. 278-307, 1971.

16. K. G. Murty, "An Algorithm For Ranking All the Assignments in Order of Increasing Cost," *Operations Research*, vol. 16, pp. 682-687, 1968.

17. T. H. C. Smith, V. Srinivasan, and G. L. Thompson, "Computational Performance of Three Subtour Elimination Algorithms for Solving Asymmetric Traveling Salesman Problems/' *Annals of Discrete Mathematics*, vol. l, pp. 495-506, 1977.

18. R. M. Karp, "A Patching Algorithm for the Nonsymmetric Traveling Salesman Problem," *SI AM Journal on Computing*, vol. 8, pp. 561-573, 1984.

19. R. Rettberg and R. Thomas, "Contention is No Obstacle to Shared-Memory Multiprocessing/* *Communications of the ACM*, vol. 29, pp. 1202-1212, 1986.

augmented transistion cost matrix

	0	1	2	3	4	5	6	7	8	9
0	oo	53	67	0	11	38	0	9	42	25
1	39	0	73	85	14	68	90	87	41	16
2	67	13	0	44	64	59	1	61	65	73
3	18	39	8	0	65	23	20	68	76	60
4	94	92	92	90	oo	70	92	73	83	45
5	44	70	57	82	31	0	1	87	25	11
6	80	43	84	73	86	82	0	69	36	3
7	23	47	12	1	75	52	83	0	77	9
8	4	64	48	38	59	87	96	59	17	5
9	6	94	88	34	23	39	17	76	56	12

benefits associated with job processing

0	1	2	3	4	5	6	7	8	9
0	14	26	14	10	17	24	25	21	28

Figure 1 An example scheduling scenario with nine jobs plus an artificial job 0 to designate production startup and shutdown

objective: maximize the sum of benefits
minus the sum of costs

optimal schedule : (0, 7, 3, 2, 6, 9, 4, 5, 8)

jobs not processed : 1

optimal objective function value : 21

objective: maximize benefits such
that costs do not exceed 120

optimal schedule : (0, 4,7, 3, 2, 6, 9)

jobs not processed : 1, 5, 8

optimal objective function value : 120

objective : minimize production costs
without regard for benefits

optimal schedule : (0, 4, 9)

jobs not processed : 1, 2, 3, 5, 6, 7, 8

optimal objective function value : 79

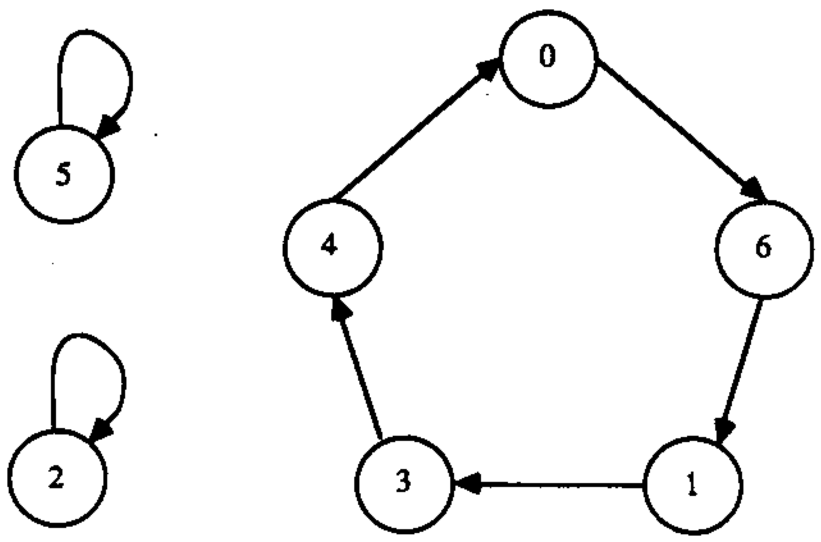
objective: minimize costs while achieving
80% of available benefits (143)

optimal schedule : (0, 7, 3, 2, 6, 9, 4, 8)

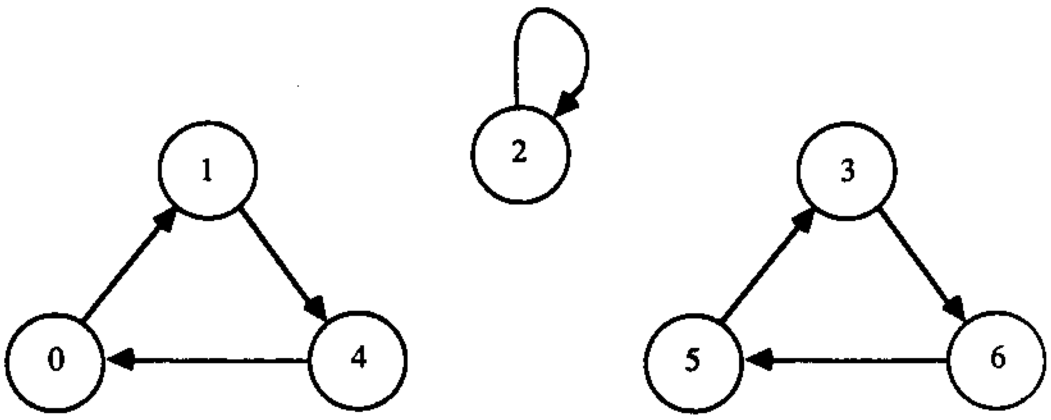
jobs not processed : 1,5

optimal objective function value : 148

Figure 2 Optimal production sequences for each of
the four scheduling objectives



graph representing a feasible pnxluction sequence



graph representing an infeasible production sequence

Figure 3 Graphs representing feasible and infeasible production sequences

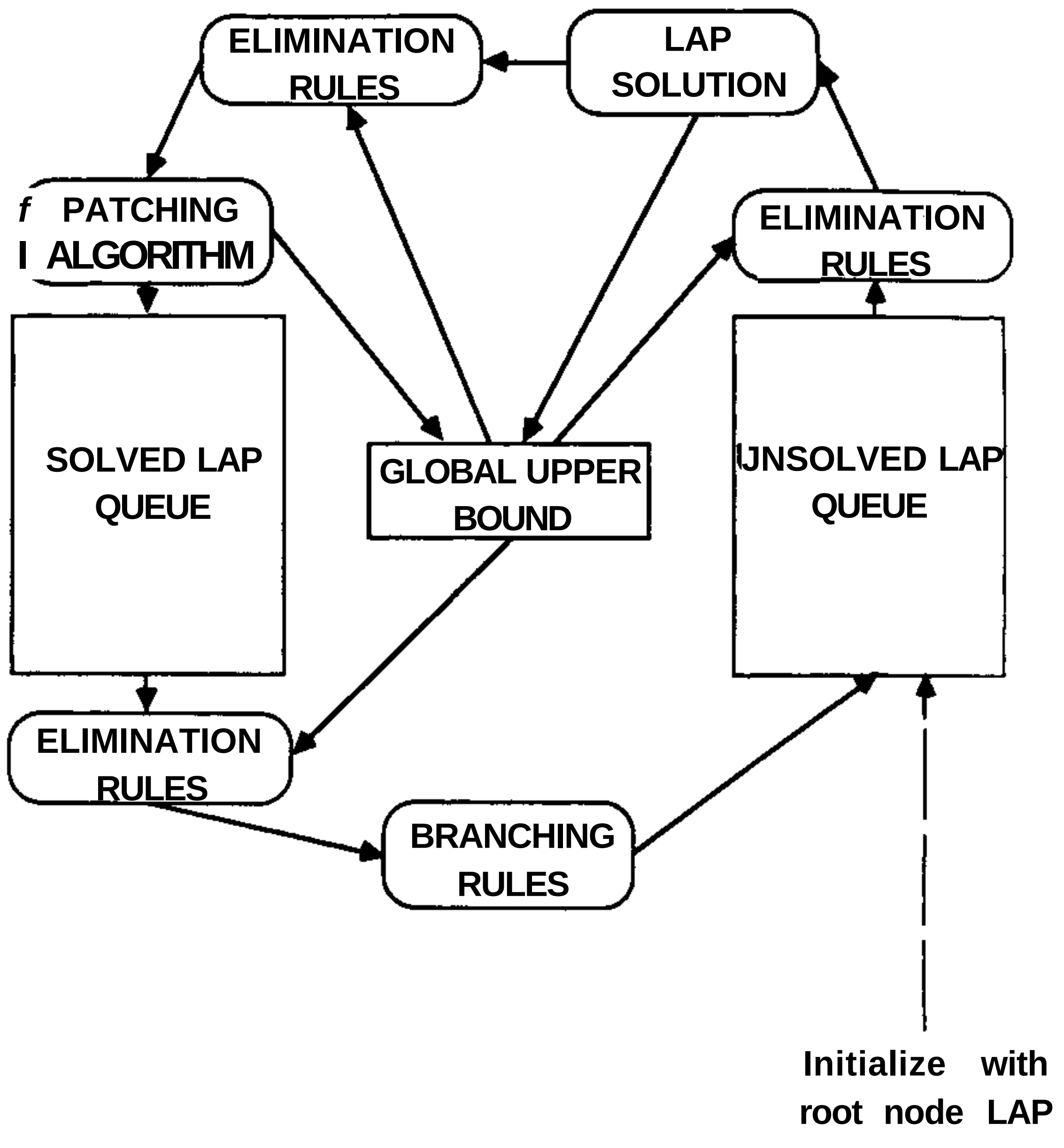


Figure 4 Data flow diagram for parallel PCTSP branch and bound algorithm

```
(1) if (unsolved LAP queue not empty) then
        remove an LAP; from the unsolved queue using best bound first selection,
        use elimination rules to delete LAPj if possible (if LAPi eliminated, go to (1)).
        solve LAPi using Lagrangian lower bounding technique,
        replace global upper bound if possible (if replaced, delete LAPj and go to (1)).
        use elimination rules to delete LAP} if possible (if LAPj eliminated, go to (1)).
        apply patching algorithm to LAPj and replace global upper bound if possible,
        place LAPj on solved LAP queue,
        goto(1).
    end if.
(2) if (solved LAP queue not empty) then
        remove an LAP^ from the solved queue using best bound first selection,
        use elimination rules to delete LAPfc if possible (if LAP^ eliminated, go to (1)).
        apply branching rules to LAP^ to generate new assignment problems,
        place new assignment problems on unsolved LAP queue,
        delete LAPfc.
        goto(1).
    end if.
(3) mark processor as idle.
    loop
        if either queue becomes nonempty, mark processor as working and go to (1).
        if all processors become idle, terminate execution.
    end loop.
```

Figure 5 Control algorithm used by all processors participating in the branch and bound algorithm

scalar qualities:

lb	lower bound value obtained by solving a Lagrangean assignment problem
X	near optimal value of Lagrangian dual calculated by the lower bounding procedure
ub	upper bound value obtained by the extended patching algorithm
nloops	number of loops present in lower bound solution
nsubtours	number of subtours present in lower bound solution
Ecard	number of exclude arcs contained in LAP
Icard	number of include arcs contained in LAP

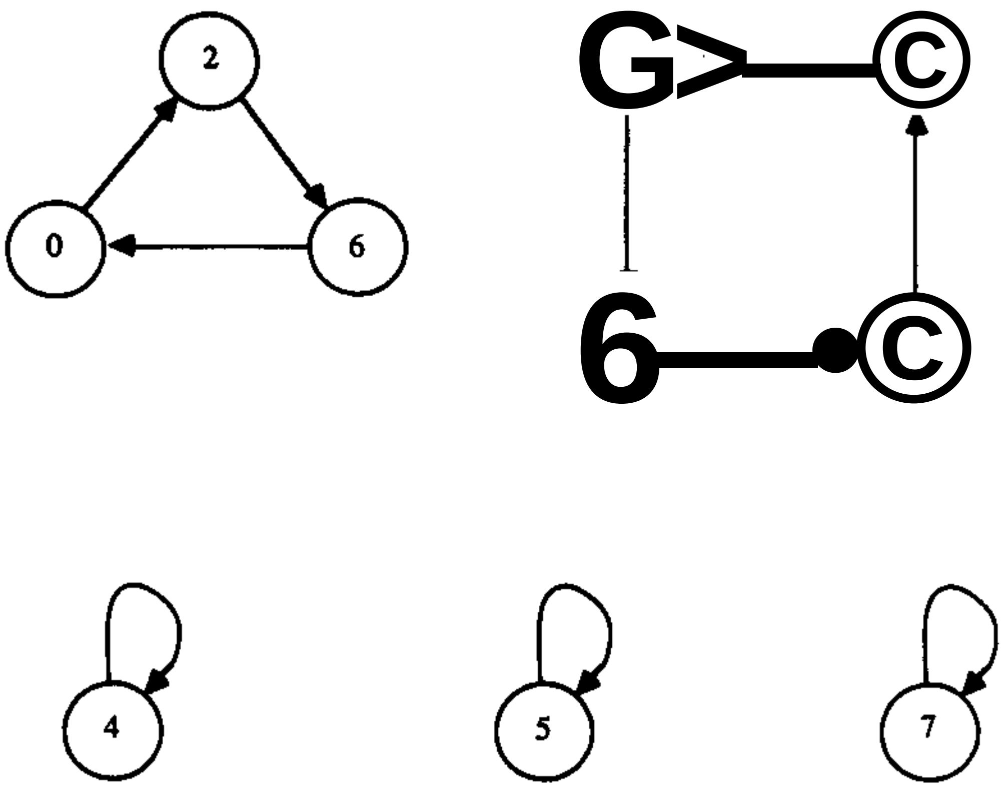
non-scalar quanties:

f	fj is the successor of vertex i (i.e. x^f.^{1} = 1)
f	fj is the predecessor of vertex j (i.e. x£j = 1)
u	UJ is the dual variable associated with row i
v	VJ is the dual variable associated with column j
loops	loops contains those vertices which participate in a loop
I	set of include arcs (i.e. xy = 1 if arc (ij) is a member of I)
E	set of exclude arcs (i.e. XJJ = 0 if arc (ij) is a member of E)

comments:

An LAP completely describes a node of the search tree for all operations used during the branch and bound algorithm.

Figure 6 Information present in an LAP



$$\Sigma p_i - P_{min} = 10$$

city prizes (benefits)

0	1	2	3	4	5	6	7	8	9
0	7	6	4	3	6	13	5	2	11

Figure 7 Graph of lower bound solution that violates subtour and prize constraints

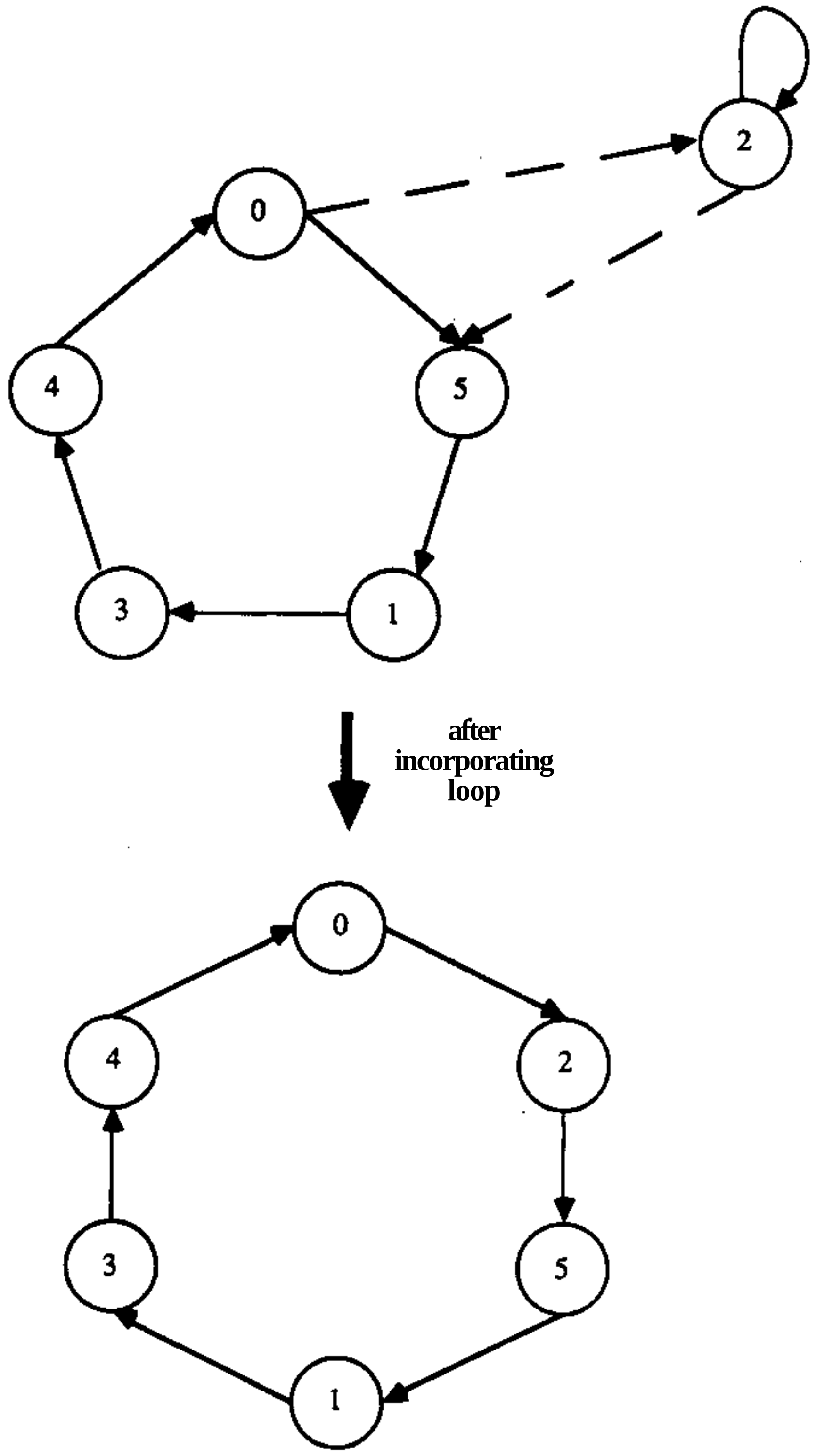


Figure 8 Incorporating a loop into a tour. One of the operations of the extended patching algorithm.

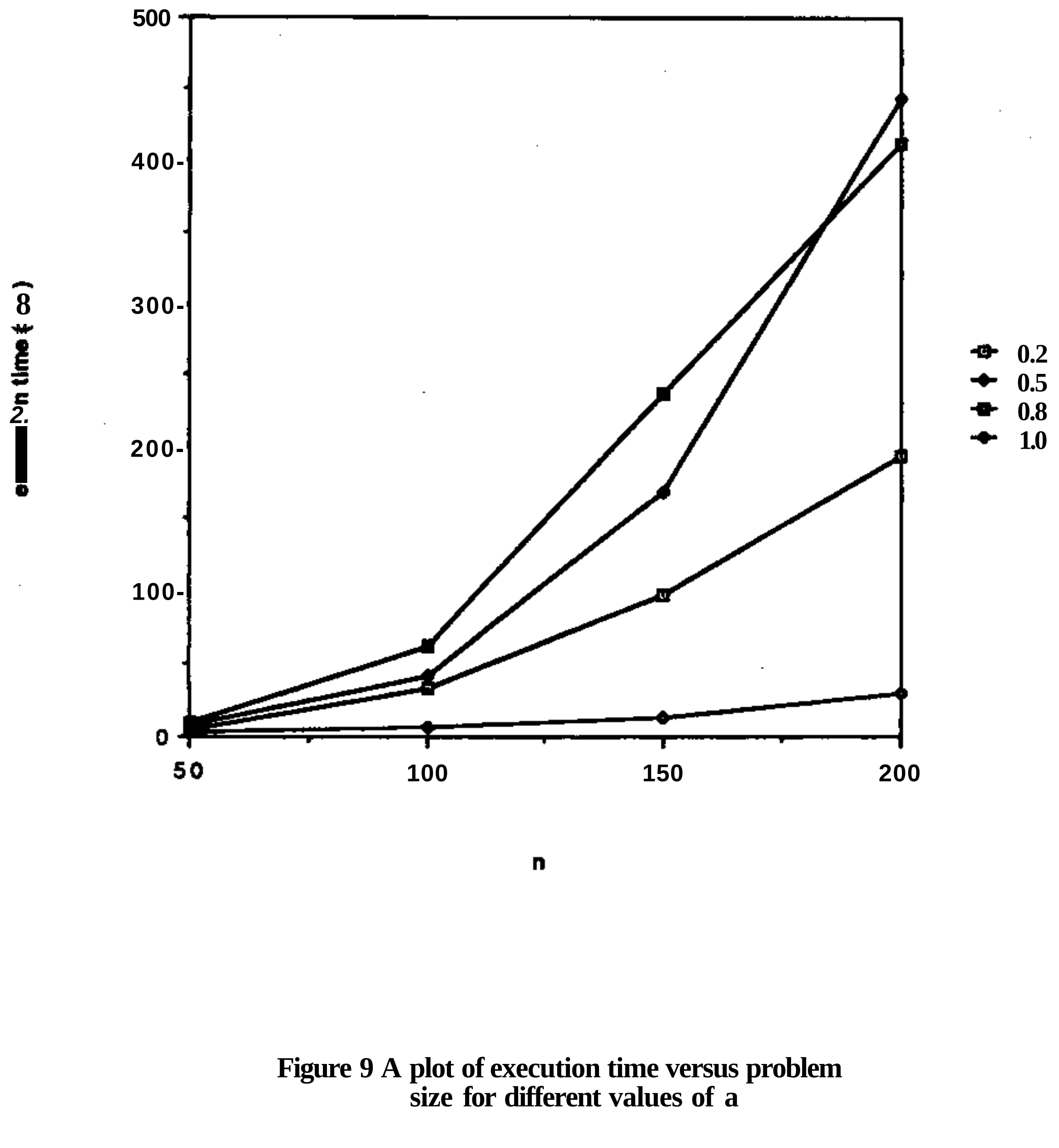


Figure 9 A plot of execution time versus problem size for different values of a

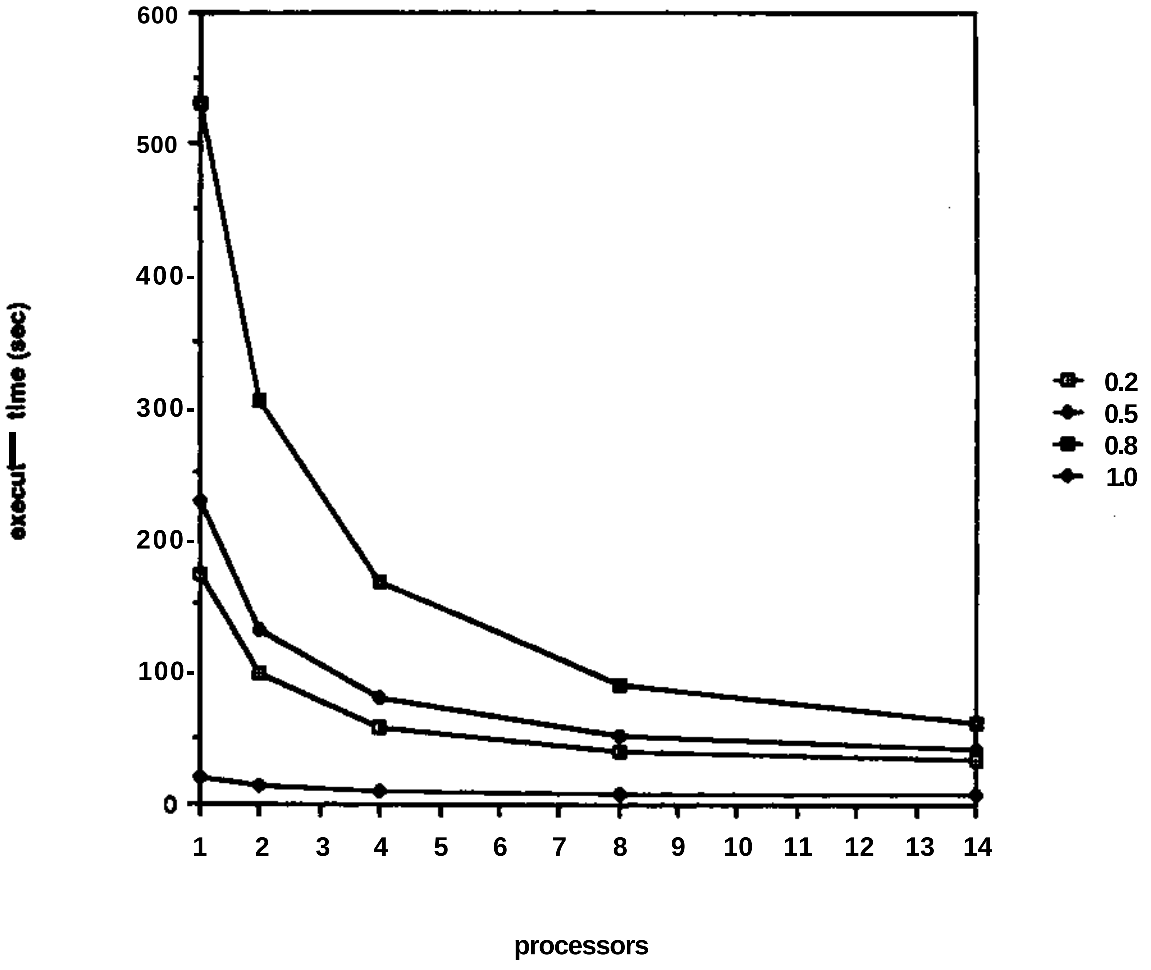


Figure 10 A plot of execution rime versus number of processors for different values of a

Table 1 Summary of scheduling objectives

problem number	objective	formulation	literature name	literature references
1	maximize the sum of benefits minus the sum of costs	equations (3-7)	asymmetric traveling salesman problem (ATSP)	references [3,4,13,14,15,18]
2	minimize production costs without regard for benefits	equations (4-7,8)	asymmetric traveling salesman problem (ATSP)	references (3,4,13,14,15,18]
3	maximize benefits such that costs do not exceed some fixed amount	equations (4-7,9,10)	orienteering problem	references [5,6,7]
4	minimize costs while attaining some level of benefits	equations (4-8,11)	prize collecting traveling salesman problem (PCTSP)	references [7,8]

Table 2t

a = 0.20

n	cases	time (avg/std)	nodes (avg/std)	ratio (avg/std)
50	10	5.15/ 2.03	75.80/43.85	1.145/0.140
100	10	32.33/15.76	377.20/315.71	1.114/0.099
150	10	98.17/33.88	788.00/545.88	1.076/0.048
200	10	187.98/86.70	797.00/751.33	1.056/0.015

$\alpha =$ 0.50

n	cases	time (avg/std)	nodes (avg/std)	ratio (avg/std)
50	10	8.48/ 4.53	225.00/249.43	1.081/0.030
100	10	46.54/14.83	415.40/227.25	1.044/0.020
150	10	183.94/92.99	1436.20/944.68	1.033/0.010
200	10	461.32/417.50	2494.10/2794.41	1.026/0.014

a = 0.80

n	cases	time (avg/std)	nodes (avg/std)	ratio (avg/std)
50	10	11.61/ 4.77	360.30/269.50	1.060/0.025
100	10	67.37/27.65	1056.40/458.11	1.024/0.009
150	10	254.80/235.57	2147.90/2364.26	1.011/0.003
200	10	477.05/243.34	1505.60/1181.03	1.015/0.009

a > 1.00

n	cases	time (avg/std)	nodes (avg/std)	ratio (avg/std)
50	10	3.19/ 0.95	75.30/47.16	1.026/0.018
100	10	6.83/ 2.84	88.10/91.57	1.006/0.005
150	10	12.86/ 3.09	78.00/56.32	1.005/0.004
200	10	28.07/ 6.77	155.40/94.12	1.005/0.003

t data collected on a 14 processor BBN Butterfly Plus

Table 3†

α = **0.20**

processors	cases	time (avg/std)	nodes (avg/std)
1	10	186.09/111.73	328.30/286.03
2	10	104.89/63.84	344.00/277.41
4	10	57.58/30.76	344.80/282.70
8	10	40.04/19.37	370.80/311.85
14	10	32.33/15.76	377.20/315.71

α = **0.50**

processors	cases	time (avg/std)	nodes (avg/std)
1	10	268.22/123.64	285.30/209.78
2	10	149.39/71.67	295.70/204.95
4	10	88.81/41.19	302.70/204.29
8	10	57.27/20.24	337.10/202.07
14	10	46.54/14.83	415.40/227.25

α = 0.80

processors	cases	time (avg/std)	nodes (avg/std)
1	10	616.61/339.87	1030.60/479.76
2	10	338.27/186.72	1031.10/479.87
4	10	179.26/94.96	1030.80/480.08
8	10	98.36/48.19	1032.30/479.81
14	10	67.37/27.65	1056.40/458.11

α = 1.00

processors	cases	time (avg/std)	nodes (avg/std)
1	10	19.76/ 17.16	75.40/80.25
2	10	12.53/ 10.02	77.00/82.86
4	10	8.72/ 5.23	76.90/81.60
8	10	6.84/ 3.21	81.80/87.91
14	10	6.83/ 2.84	88.10/91.57

† data collected on a 14 processor BBN Butterfly Plus