Application of a Parallel Traveling Salesman Problem Algorithm to No-Wait Flowshop Scheduling

by

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

EDRC 06-51-89 >J>

Application of a Parallel Traveling Salesman Problem Algorithm to No-Wait Flowshop Scheduling

/. F. Peknyt

D. L. MiUerX

G. J. McRaet

ABSTRACT

Intermediate storage utilization provides a key constraint in the design of multi-step batch process production schedules. In some cases, unstable intermediate products must be processed without delay at any production step so that intermediate storage may not be used. In a flowshop, such a no-wait scenario can be reduced to a traveling salesman problem. This paper details the application of a parallel branch and bound algorithm for the traveling salesman problem to solve the no-wait flowshop scheduling problem. The algorithm is based on an assignment problem lower bounding technique, branching rules that partition the search tree, a patching algorithm based upper bounding technique, and a directed hamiltonian circuit reduction technique. The algorithm is implemented on a BBN Butterfly Plus shared memory multiprocessor. Computational results are presented to demonstrate the effectiveness of the parallel algorithm.

Key Words: Flowshop Scheduling, Parallel Algorithm, Traveling Salesman Problem

October 31, 1988

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

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

Application of a Parallel Traveling Salesman Problem Algorithm to No-Wait Flowshop Scheduling

7. *F. Peknyt*
D. L. MiUeri
G. J. McRaet

Introduction

In many scheduling situations multistage jobs consist of a sequence of operations which must be performed on a set of machines. When all jobs use the same set of machines in the same order, the scheduling situation is known as a flowshop. Flowshop scheduling has been studied under a variety of assumptions (see [1,2]). Flowshop scheduling schemes may be classified according to the assumptions made about storage available for partially processed jobs. Intermediate storage assumptions fall in one of four **categories:**

(1) **unlimited intermediate storage** [3,4,5,6,7] - Any number of partially processed jobs may be stored while awaiting further processing.

(2) **finite intermediate storage** [8,9,10] - Only a finite number of partially processed jobs may be stored while awaiting further processing. In the limiting cases of a large amount or a small amount of intermediate storage, this situation can give results similar to the unlimited intermediate storage situation or the no-wait processing situation, respectively.

(3) **no intermediate storage** [11,12] - The production machines serve as storage units. While a machine is storing a job that it has completed processing, further processing of other jobs is blocked.

(4) **no-wait processing** [13,14,15,16] - Partially completed jobs must be processed without delay. Under this constraint, job processing is uninterrupted until the job is completed. Total job residence time in the flowshop is the sum of the processing times on the individual machines.

The classification of flowshop scheduling situations based on intermediate storage assumptions is somewhat artificial since many practical flowshop situations present a mixed storage environment. For example, unlimited intermediate storage may be available to some machines in a flowshop while the no-wan processing condition applies to some other segment of the flowshop.

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

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

The algorithm presented in this paper is applicable to fiowshop scheduling with the no-wait processing condition. No-wait processing is necessary when intermediate storage is unavailable or when **partially processed** jobs cannot be delayed. Consider the following examples where no-wait flowshop **scheduling is an appropriate** model:

rolling mill processing [15] - Hot ingots are to be rolled to specified thicknesses using a series of rolling mills. The rolling mills are organized as a flowshop with each successive mill reducing an ingot's thickness. Processing the ingots at high temperature is critical both to product quality and minimizing rolling mill wear. Heating the ingots is a time consuming operation due to large thermal masses. Because the ingots cannot be allowed to cool between mill rollings, no-wait processing is a necessity. An optimal schedule specifies the order in which the ingots should pass through the mills to minimize total processing time.

computer pipeline scheduling [14] - Computers with pipeline processing capability have become quite common (i.e. Cray series, RISC chip based computers, etc.). The computer pipeline functions much like an automobile assembly line where processors in the pipeline perform tasks that contribute to job completion. When a pipeline processor finishes its task, a job is passed to the next processor in the pipeline. If jobs are to be stored between processors in the pipeline, buffers must be available. In order to match the processing speeds of the pipeline, buffers must be made of fast expensive memory chips. In addition, management of partially finished jobs in buffers can become quite complicated. For these reasons computational jobs are to be scheduled through the pipeline using the no-wait processing condition. The pipeline manager seeks to minimize the makespan of a set of computational jobs. In the interests of management efficiency, the execution time of the scheduling algorithm must be small with respect to the makespan of the computational jobs.

chemical processing [8] - Multistage manufacturing processes for a certain family of specialty chemicals (e.g. fluorocarbon polymers) requires intermediates that are unstable. Once the intermediates are produced, they must be processed immediately. The same set of batch and semi-continuous process equipment is used to manufacture all members of the chemical family. A schedule must be determined ihat fills orders placed for different members of the chemical family in as little time as possible.

Figure 1 provides a numerical example of a flowshop scheduling situation with the no-wait processing condition. Eight jobs must be processed using four machines in as little time as possible. An clement *tij* of the processing time matrix describes the amount of time job i requires on machine j. Figure 2 shows a Gantt chart of an optimal makespan schedule. The following sections will describe the algorithm used to calculate the optimal schedule shown in Figure 2.

Most flowshop scheduling research has concentrated on developing optimal or near optimal schedules that minimize the time required to process all jobs (makespan). Other possible scheduling objectives include minimizing machine idle time, minimizing the mean time required to process all jobs, minimizing maximum or mean tardiness for those jobs involving deadlines, or minimizing the cost of processing all jobs (see [17]). Each of these scheduling objectives is appropriate in certain production environments.

The branch and bound algorithm presented in the following sections is designed to minimize production costs in a flowshop with no-wait processing. Production costs are a function of the following attributes:

(a) machine idle time - Each unit of idle time on every machine incurs a cost.

(b) production line changeover costs - A changeover cost is incurred for switching machines between jobs. The changeover cost is modeled as a function of consecutive jobs passed through the flowshop production line.

(c) production line startup costs - A cost is incurred for starting production up to process ihe first job. This startup cost is a function of the job that will be processed first.

(d) production line termination costs - A cost is incurred for terminating production after the last job is processed. This termination cost is a function of the job that is processed last.

(e) production omission penalties - A flowshop production schedule extends over a certain time frame. Of the jobs available for production, a certain number may be delayed until the next scheduling time frame. Production omission penalties are the cost of not processing a job during the current scheduling time frame.

Costs can be a measure of machine idle time, dollar expenditure, makespan, or any other resource consumed by production. In the next section, a formulation is given that accounts for these production costs.

Previous research reported in [13,14,15] has shown the equivalence between no-wan (lowshop scheduling and the traveling salesman problem (TSP). The derivation shown below generalizes this equivalence to account for cost attributes (a-e) as well as multiple flowshop production lines. Because the problems are equivalent, an effective algorithm for no-wait flowshop scheduling requires an effective algorithm for the TSP. The algorithm presented below is designed for the asymmetric traveling salesman problem (ATSP). The asymmetric label refers to the asymmetry of the TSP cost matrix (i.e. the intercity distances). The algorithm presented below is ineffective on no-wait flowshop scheduling problems that give rise to symmetric cost matrices and certain specially structured asymmetric cost matrices (see [18]). Note that [19] shows how a special case of job-shop scheduling with no intermediate storage can be reduced to a TSP. The work reported in [20] shows how a more general case of job-shop scheduling with no intermediate storage can be reduced to a collection of TSPs.

Problem Formulation

A set of N jobs is available for processing at time zero. Each multistage job is composed of M operations that must be conducted on M distinct machines in sequence. Job operations are not preemptable. A solution to the flowshop problem specifies the order in which jobs are processed on each machine. The no-wait processing condition forces each machine to process the jobs in the same order (see [13]). The flowshop is composed of P identical production lines. Each production line is composed of M distinct machines on which job operations are conducted. Each production line is capable of conducting all operations associated with a job. Each job is processed on a single production line.

Each of the N jobs is represented by a city in the TSP formulation. The N jobs (cities) are denoted by the index set {1,...,N}. Each of the P production lines is also represented by a city in the TSP formulation. The P production lines (cities) are denoted by the index set {N+ 1,...,N+ P}. The total number of cities is denoted n. The processing time matrix t= (/,;) of size N by M indicates the amount of processing time job i requires on machine j in any of the production lines.

The no-wait condition guarantees that once a job commences there is no delay in processing. In order to prevent a conflict between job i and a subsequent job j processed on the same production line, job j must be delayed D(iJ) time units between the completion of job i on the first machine and the start of job j. The delay is a function of the processing time matrix as follows (see [13]):

$$D(i,j) = \max_{2 \le m \le M} \left[\sum_{k=2}^{m} t_{ik} - \sum_{k=1}^{m-1} t_{jk} , 0 \right] \quad (1)$$

The delay between jobs i and j implies an idle time $I_m(iJ)$ on machine m in a production line. The idle time may be computed as a function of the delay as follows (see [13]):

$$I_m(i,j) = D(i,j) + \sum_{k=1}^{m-1} t_{jk} - \sum_{k=2}^{m} t_{ik} \quad (2)$$

Given the definition for idle time, the TSP formulation of the no-wait flowshop scheduling problem becomes:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \quad (3)$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad i = 1,...,n \quad (4)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad j = 1,...,n \quad (5)$$

Z Z *XijZ\S\-l,* for *each* Sc(1,...,n) (6)

^€{0,1}, *for each i,je{l,...,n)* (7)

where,

$$c_{ij} = \sigma_j + \sum_{k=1}^{M} \omega_k Z_{ij}^{k-1} \quad \text{for } j \in \{1,...,N\},\ i \in \{N+1,...,N+P\} \tag{8}$$

$$c_{ij} = \kappa_{ij} + \sum_{k=2}^{M} I_k \ll k Ik(iJ), \quad \text{for } I\ J\ E\ \{1,...,N\},\ i^*\ y \tag{9}$$

$$c_{ji} = \tau_i \quad \text{for } i \in \{1,...,N\},\ j \in \{N+1,...,N+P\} \tag{10}$$

$$\ll^* = \diamond, \quad \text{for } i \in \{1,...,N) \tag{11}$$

$$c^* = X, \quad \text{for } i E \{tf+1,...,JV+P\} \tag{12}$$

and,

(0* = *cost of a unit of idle time on the k-th machine in any non-idle production line*

Oj=cost of starting production with job j

<ij=cost of producing job j after job i on a non-idle production line

Xi=cost of terminating production with job i

oncost of not producing job i within the time frame of the model

\=cost of an idle production line

Figure 3 indicates the structure of the TSP cost matrix. Without loss of generality, the elements of the TSP cost matrix are assumed to be integer. Equations (1-12) completely define the no-wait flowshop problem in terms of a TSP. If $x_u = 1$ for *ie* {1,...,N} then job i is not processed within the time frame of the scheduling model. If $x_{tf} = 1$ for *ije* {1....JV} and *i * j* then job j follows job i in a production sequence. Job j is processed on the same production line as job i. If $x_{tj} = 1$ for *ie {N + 1....N - P)* and *je* {l,...,Af} then job j is the first job processed on production line i-N. If $JC_{i;} = 1$ for *i*E (1.....v) and *je [N+* 1,...,# + />} then job i is the last job processed on the production line used to process job i. If $X_{jy} = 1$ for *ije* {Af+ *l,...9N + P)* then production line j is not used and production line i assumes the responsibilities of production line j. If *#= 1 for *is [N+l,...,N + P)* then production line i is not used to process any jobs. Equations (1-12) constitute a generalization of the results given in [13]. Equations (1-12) can be specialized to account for common no-wait flowshop objectives. Consider the following specializations:

(i) **minimize makespan** - set (0,=0 for all iE {1,...,M-1}, 0)^=1, <Jy=O for all $JE\{1,...,N\}$, $K_{I;} = 0$ for all i.JE {1,...,N}, !,;=() for all JE {1,...,N}, o_t=<» for all iE {1.....N), and X=«> to minimize the total time required to process all N jobs on P production lines. The makespan minimization problem shown in Figure 1 was solved as a special case of equations (1-12) for P= 1 production line and N= 8 jobs.

(ii) **minimize machine idle time** - set 0)^=1 for all iE{l,...,M), *Gj=Q* for all $JE\{1,...,N)$, $ic_{iy} = 0$ for all ijE{l,...,N}, T_{4}=0 for all $JE\{1,...,N)$, O_4 = « for all iE{l,...,N}, and X=» to minimize the sum of

idle time on all machines in the flowshop.

An Exact Parallel ATSP Algorithm

The parallel ATSP algorithm used to solve equations (1-12) is an extension of the algorithm reported in [21]. The discussion below summarizes the basic algorithm and describes the algorithm extensions in detail.

Processors operate in the framework defined by the processor shop model shown in Figure 4. The processor shop model consists of a collection of data queues through which processors communicate. Processors perform operations on items removed from the data queues. The operations performed by the processors are components of the branch and bound method to TSP solution. The assignment problem data type (AP) referenced in Figure 4 encapsulates all information required by the algorithm components. When given a choice of items to remove from a data queue, processors choose the AP with the best chance of yielding an optimal TSP solution. Figure 5 shows the control algorithm each processor uses when operating in the dataflow environment of Figure 4. The combined action of all processors executing the control algorithm of Figure 5 locates an optimal TSP solution. The global upper bound data queue stores the best TSP solution known at any time during algorithm execution. Any valid TSP solution discovered during algorithm execution is used to update the global upper bound queue. Upon algorithm termination the global upper bound data queue stores an optimal TSP solution.

Equations (3-5,7) define the assignment problem. The assignment problem is a well known relaxation of the TSP (see [22]). A solution to the assignment problem does not necessarily satisfy the constraints imposed by equation (6) of the TSP formulation. In order to visualize the relationship between an assignment problem solution and the corresponding TSP solution consider the graph G = (V,A) where Vs {l,...,n} and A= {(i,j) | i,j € V}. The arcs of G correspond to the integer variables x in equations (3-7). Variable x_{ij} corresponds to arc (i,j). Figure 6 illustrates a subgraph of G corresponding to a TSP solution containing 7 cities. An arc (i,j) is placed between vertices i and j in Figure 6 only if x,y= 1. As Figure 6 shows a TSP solution requires that all cities participate in either a directed cycle or loopst. In the TSP literature a single directed cycle is known as a tour. Figure 7 illustrates a subgraph corresponding to an assignment problem solution containing 7 cities. Figure 7 shows that an assignment problem solution requires that all cities participate in loops or directed cycles of two or more cities. Multiple directed cycles of two or more cities are known as subtours. Equations (3-7) represent a slightly more general version of the TSP than is typically found in the literature. Equations (3-7) permit loops in the TSP solution whereas normal TSP formulations forbid loops (see [22]). Loops can be forbidden in the solution to equations (3-7) by setting all the diagonal elements of the cost matrix to

t A loop is t directed cycle involving only one vertex.

infinity. **Permitting loops** in **the** TSP solution provides for generality in expressing no-wait flowshop **problems. Loops present** in **the** optimal TSP solution indicate those jobs not processed in the time frame **of the scheduling** modcL

There is a one to one correspondence between the AP manipulated in Figure 4 and nodes in the branch and **bound** tree. By encapsulating all node information in an AP, processors can operate independently once an AP has been removed from a data queue. Processors coordinate queue insertion and removal to avoid corrupting the queue data structures. An AP is classified as either solved or unsolved. The lower bounding procedure has been applied to solved AP but not to unsolved AP. Each solved AP has an associated lower bound calculated from solution to an assignment problem. When choosing between problems to remove from the data queues, a problem with the least lower bound is chosen first. Such a selection rule is consistent with the philosophy of always choosing APs that have the best chance of leading to an optimal TSP solution. Unsolved APs inherent their parent's lower bound for purposes of queue removal. An unsolved AP always has a solved AP as a parent.

Unsolved APs are created from solved AP using the branching rules. The branching rules act according to the assignment problem solution contained in an AP. The **ATSP** algorithm uses a modification of the branching rules reported in [23,24,25,26]. The branching rule modifications involve ignoring loops present in an assignment problem solution. The branching rules create two or more unsolved AP for every solved AP. The branching rules are enforced by modifying the cost matrix of equation (3). Each AP has its own unique cost matrix modifications assigned by the branching rules. The modifications involve replacing some elements of the cost matrix with infinity to forbid certain subtours in an assignment problem solution. A child AP inherents all cost matrix modifications of a parent AP. Computational experience has shown that the number of matrix modifications required at any node is quite small (i.e. O(n)) relative to the number of cost matrix elements (i.e. $O(n^2)$). Each AP stores its own unique set of modifications as a list instead of modifying the cost matrix directly. Storing the modifications as a list eliminates the need for multiple copies of the cost matrix.

A solved AP is created from an unsolved AP using the lower bounding procedure. The lower bounding **procedure** uses the assignment problem solution technique described in [27] to solve the root node **AP. The parallel** assignment problem solution technique has computational complexity $O(n^3)$. A parametric assignment problem solution procedure is applied to all other nodes in the branch and bound tree. The parametric procedure uses the assignment problem solution of the parent node as well as branching rule information to solve the assignment problem associated with a child node. The parametric technique has computational complexity $O(n^2)$.

The patching algorithm is an upper bounding heuristic that splices subtours together to form a tour. By providing near optimal TSP solutions, the patching algorithm may reduce the amount of work

processors **need to** perform **(see** [21]). A complete description of the patching algorithm may be found in [28].

The elimination rules in combination with the lower bounds and the global upper bound prevent the branch and bound algorithm from being an exhaustive search of the solution space. Whenever the lower bound associated with an AP equals or exceeds the global upper, the AP is eliminated from further consideration. This elimination rule allows the branch and bound algorithm to avoid explicit consideration of the vast majority of the (n-1)! TSP solutions. An AP is also eliminated from consideration when the associated assignment problem solution represents a valid TSP solution.

An ATSP Reduction Technique

The constraint matrix implied by the linear assignment problem formulation (equations (3-5,7)) is totally unimodular (see [29]). Thus the integrality constraint (equation (7)) may be relaxed to yield the following equivalent linear program:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}X_{ij} \qquad (13)$$

$$\sum_{i \neq i} x_{iy} = 1, \quad i = 1,\ldots,n \qquad (14)$$

$$\sum_{j=1}^{n} x_{xy} = i, \quad y = i,\ldots,n \qquad (15)$$

$$x_{ij} \geq 0, \quad i,j = 1,\ldots,n \qquad (16)$$

The dual problem associated with the primal problem represented by equations (13-16) can be formulated as:

$$\max \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j; \qquad (6)$$

$$c_{ij} \geq u_i + v_j, \quad i,j = 1,\ldots,n \qquad (7)$$

The optimal objective function value of the dual problem is equal to the optimal objective function value of **the primal** problem (see [30]). The lower bounding procedure used in the ATSP algorithm determines optimal dual variable values u_i, v_j at each node of the branch and bound tree using a cost matrix $\bar{c}$ that has been modified by the branching rules. The optimal dual variable values can be used to generate a directed graph $\bar{G} = (\bar{V}, \bar{A})$ where $\bar{V} = \{1,\ldots,n\}$ and $\bar{A} = \{(ij) \mid \bar{c}_{ij} - u_i - v_j = 0 \text{ for all } i,j \in V\}$. Arc (ij) of set $\bar{A}$ indicates that x_{ij} can be equal to one in an optimal assignment problem solution. The ATSP reduction technique determines if $\bar{G}$ contains a single directed circuit plus a collection of loops. A single directed circuit on $\bar{G}$ plus loops is a valid TSP solution. When the reduction technique discovers a valid TSP solution the associated branch and bound node (AP) can be discarded from further

consideration. **The** reduction technique utilizes **a** slight modification of the exact directed hamiltonian circuit **algorithm reported** in [31]. The modified hamiltonian circuit algorithm permits loops. Whenever **the reduction technique** fails **to** find **a TSP solution, the lower bound** associated with the AP is increased to $\sum_{i=1}^{\cdot} \{\&i + \pounds v\} + 1$. In this case, the lower bound may be increased by one because the exact hamiltonian circuit algorithm guarantees that a TSP solution will not be found at a cost of $\sum_{i=1}^{A} \pounds 5; + \sum_{/=1}^{A} \bar{v}\}$. Thus the reduction technique is used both to locate TSP solutions at a given cost and to strengthen the value of lower bounds. Computational results show that the reduction technique leads to a substantial improvement in algorithm efficiency for certain classes of TSPs (see Table 3).

Computational Results

The ATSP algorithm was tested on problems ranging in size from 500 to 3500 cities with cost matrix elements drawn from a uniform distribution of integers in the range [0,p] for p= 1000 and p= 10000. The algorithm was implemented on a BBN Butterfly Plus* possessing 14 Motorola 68020/68881 processors and 56 megabytes of memory. Each Butterfly processor provides 2.5 MIPS of computational power for a total power of 35 MIPS. Additional information concerning the Butterfly architecture may be found in [32]. Additional information concerning the mapping of the ATSP algorithm onto the Butterfly architecture may be found in [21,27].

Table 1 presents execution information as a function of problem size and cost range. The table reports the average and standard deviation of the total execution time, the number of assignment problems solved, and the ratio of the optimal TSP solution value to the root node lower bound. The statistics were collected for seven problems for each size and cost range. As the cost range becomes small with respect to the number of cities the TSPs become easier to solve as measured by the number of assignment problems solved. In fact for the 3500 city problems with the [0,1000] cost range, only the root node assignment problem has to be solved. For all of these 3500 city problems the reduction technique determined an optimal solution at the root node of the branch and bound tree. The reduction technique is successful for these problems because $\vec{G}$t becomes denser as the cost range becomes small with respect to problem size. The ratio column in Table 1 shows that the assignment problem provides excellent lower bounds for randomly generated ATSPs. Table 1 also shows a high variation in both total execution time and number of assignment problem solved for a given problem size and cost range.

t **Butterfly Plus is t trademark of** Bolt, **Berwick, and Newman** Inc.

t **See the section describing** the reduction technique.

Table 2 **presents** algorithm execution statistics as a function of the number of processors and the cost **range. The same** SOO city problems were solved using a variable number of processors to generate Table 2. Thus **Table** 2 indicates the benefit of using multiple processors to solve a TSP. As Table 2 shows, 14 processor speedups for the SOO city problems range from 1.67 for the [0,1000] cost range to 6.30 for the [0,10000] cost range. The low speedups for the small cost range are attributable to the small number of assignment problems that have to be solved. Since the parallelism is concentrated at the branch and bound level, a small number of search tree nodes translates into a small amount of parallel work. With the larger cost range the number of assignment problems solved is much greater and the speedup numbers are significantly higher. Table 2 indicates that for the small cost range parallelism needs to be concentrated below the branch and bound level. Parametric assignment problem solution and the reduction technique are the two best candidates for low level parallelization. Table 2 shows that parallel branch and bound reduces both the execution time and the variation in execution times. Thus parallel branch and bound makes the overall execution times more predictable. A deleterious effect of parallel branch and bound is the increase in the amount of overall computational work as measured by the number of assignment problems solved. Part of the benefit of using multiple processors is offset by this increase in the size of the branch and bound trees.

Table 3 demonstrates the value of the reduction technique in the branch and bound algorithm. As the table shows, the reduction technique is most useful when the problem size is large with respect to the cost range. For the SOO city problems and the [0,10000] cost range the reduction technique increased the average execution time. This suggests that the reduction technique only be applied when the ratio of problem size to cost range reaches a certain threshold. This is equivalent to applying the reduction technique only when $\overline{G}$t reaches a certain threshold density.

Conclusions

The flowshop scheduling model with the no-wait processing condition has a number of applications. The no-wait flowshop model can be solved as a traveling salesman problem. An exact algorithm has been presented for traveling salesman problems with asymmetric cost matrices. Computational results show the algorithm to be quite effective on randomly generated problems. Computational results also show that concentrating parallelism at the branch and bound level is appropriate when problems have a large number of nodes in the search tree. When the search trees do not possess a large number of nodes, parallelism must be concentrated in lower level subalgorithms.

t **See the section describing reduction technique.**

References

1. K. **R. Baker,** *Introduction to. Sequencing and Scheduling,* John Wiley, New York, 1974.
2. G. V. **Reklaitis, "Review** of Scheduling of Process Operations,*' *AIChE Symp. Ser,* vol. 78:214, pp. 119-133, 1982.
3. S. M. Johnson, "Optimal Two- and Three-Stage Production Schedules With Set-up Times Included,'* *Naval Research Logistics Quarterly,* vol. 1, pp. 61-68, 1954.
4. B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan, "A General Bounding Scheme for the Permutation Flowshop Problem," *Operations Research,* vol. 26, pp. 53-67, 1978.
5. S. S. Panwalker and W. Iskander, "A Survey of Scheduling Rules," *Operations Research,* vol. 25, pp. 45-61, 1977.
6. G. B. McMahon, "Optimal Production Schedules for Flowshops," *Canad. Operational Res. Soc.* 7., vol. 7, pp. 141-151, 1969.
7. S. Ashour, "A Branch and Bound Algorithm for the Flowshop Scheduling Problem/' *AUE Trans.,* vol. 2, pp. 172-176, 1970.
8. F. C. Knopf, "Sequencing a Generalized Two-Stage Flowshop with Finite Intermediate Storage," *Computers & Chemical Engineering,* vol. 9, pp. 207-221, 1985.
9. S. K. Dutta and A. A. Cunningham, "Sequencing Two Machine Flowshops With Finite Intermediate Storage," *Management Science,* vol. 21, pp. 989-996, 1975.
10. C. H. Papadimitriou and P. C. Kanellakis, "Flowshop Scheduling With Limited Temporary Storage," /. *Assoc. Comput. Mack,* vol. 27, pp. 533-549, 1980.
11. I. Suhami and R. S. H. Mah, "An Implicit Enumeration Scheme for the Flowshop Problem With No Intermediate Storage," *Computers & Chemical Engineering,* vol. 5, pp. 83-91, 1981.
12. D. W. T. Rippin and M. Hofmeister, "The Flowshop Scheduling Problem With No Intermediate Storage," *Paper presented at AIChE Annual Meeting,* Symposium on Computer Aided Design of Batch and Semi-Continuous Processes, New Orleans, Louisiana, 1981.
13. J. N. D. Gupta, "Optimal Flowshop Schedules with No Intermediate Storage Space," *Naval Research Logistics Quarterly,* vol. 23, pp. 235-243, 1976.
14. S. S. Reddi and C. V. Ramamoorthy, "On the Flow-Shop Sequencing Problem With No Wait in Process," *Operational Research Quarterly,* vol. 23, pp. 323-331, 1972.
15. D. A. Wismer, "A Solution of the Flowshop Scheduling Problem with No Intermediate Queues," *Operations Research,* vol. 20, pp. 689-697, 1972.
16. J. M. Van Deman and K. R. Baker, "Minimizing Mean Flowtime in the Flowshop With No Intermediate Queues," *AUE Trans.,* vol. 6, pp. 28-34, 1974.
17. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals of Discrete*

Mathematics, voL 5, pp. 287-326, 1979.

18. 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, 198S.

19. J. K. Lenstra and A. H. G. Rinnooy Kan, "Some Simple Applications of the Traveling Salesman Problem," *Operational Research Quarterly,* vol. 26, pp. 717-734, 1975.

20. S. S. Reddi and C. V. Ramamoorthy, **A Scheduling Problem," *Operational Research Quarterly,* vol. 24, pp. 441-446, 1973.

21. J. F. Pekny and D. L. Miller, "A Parallel Branch and Bound Algorithm For Solving Large Asymmetric Traveling Salesman Problems," *Mathematical Programming,* 1988. submitted

22. 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.

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

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

25. 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. 1, pp. 495-506, 1977.

26. 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.

27. J. F. Pekny, D. L. Miller, E. Balas, and P. Toth, "A Parallel Shortest Augmenting Path Algorithm for Fully Dense Linear Assignment Problems," *Department of Chemical Engineering,* Carnegie Mellon University, Pittsburgh PA 15213. working paper

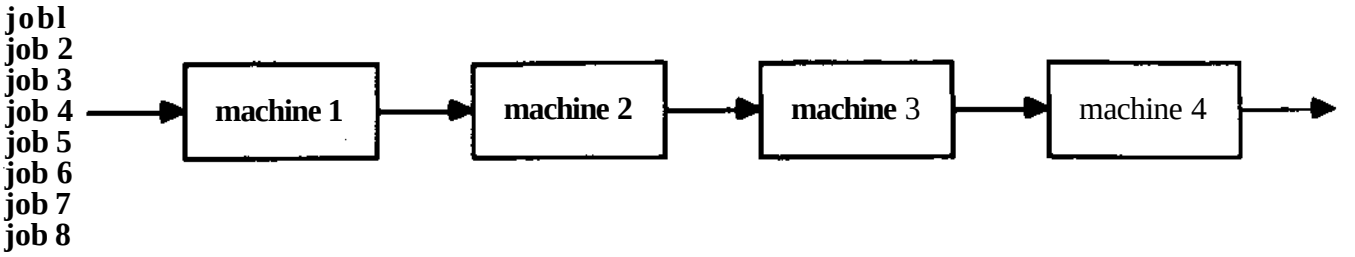
28. R. M. Karp, "A Patching Algorithm for the Nonsymmetric Traveling Salesman Problem," *SIAM Journal on Computing,* vol. 8, pp. 561-573, 1984.

29. A. Schrijver, *Theory of Linear and Integer Programming,* John Wiley & Sons, New York, 1986.

30. G. B. Dantzig, *Linear Programming and Extensions,* Princeton Univ. Press, Princeton, NJ, 1963.

31. D. Stodolsky, J. F. Pekny, and D. L. Miller, 4<An Exact Algorithm for Finding Hamiltoman Circuits in Directed Graphs," *Engineering Design Research Center,* Carnegie Mellon University. Putsburgh PA 15213. working paper

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



	machine 1	machine 2	machine 3	machine 4
jobl	9	9	7	8
job 2	8	7	5	4
job 3	2	4	5	1
job 4	6	2	9	3
job 5	8	7	7	1
job 6	9	2	8	0
job 7	8	2	4	6
job 8	0	9	3	4

processing time matrix

Figure 1 An example flowshop with 8 jobs and 4 machines subject to the no-wait processing condition

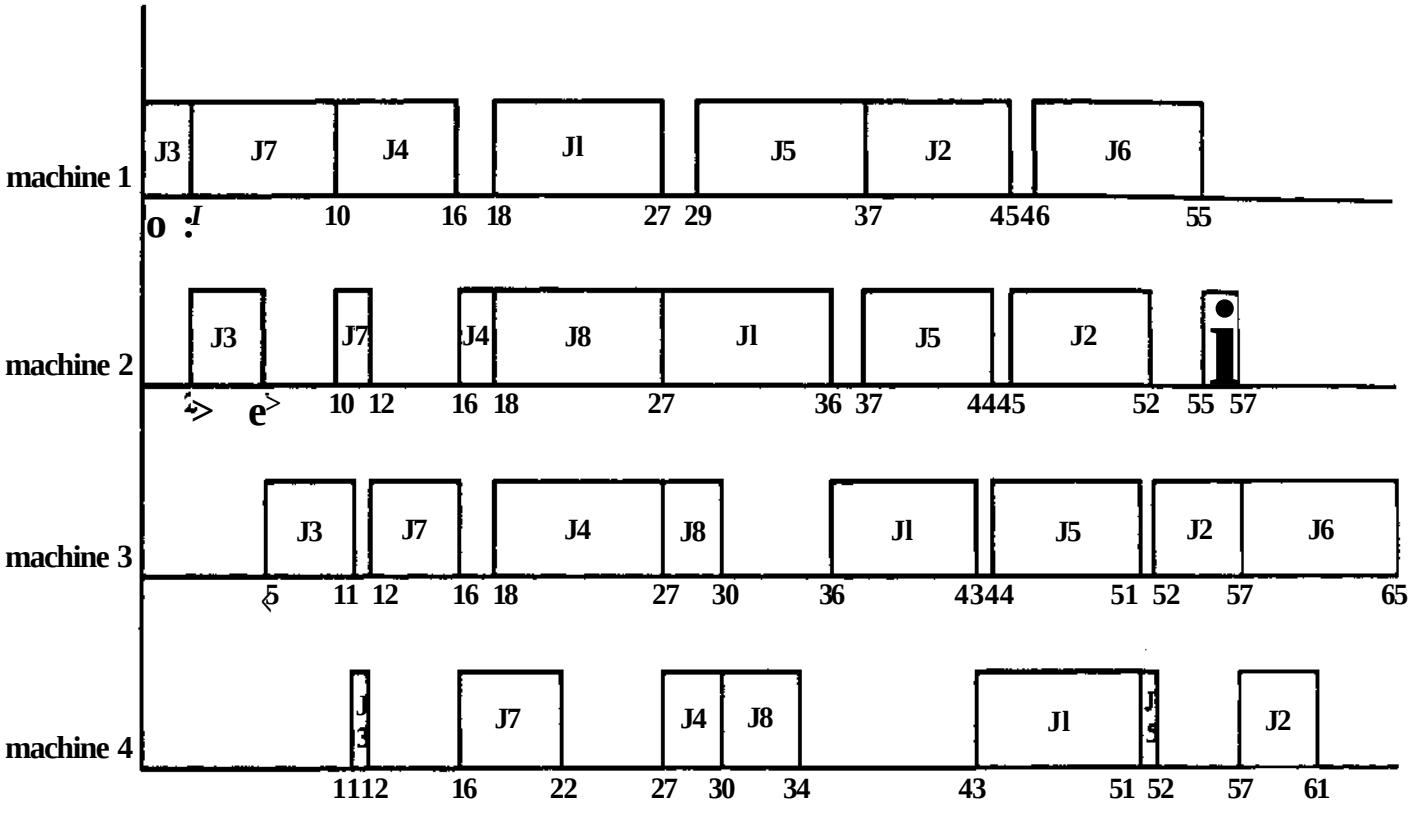


Figure 2 Gantt chart of an optimal makespan solution to the 4 machine and 8 job example shown in Figure 1.

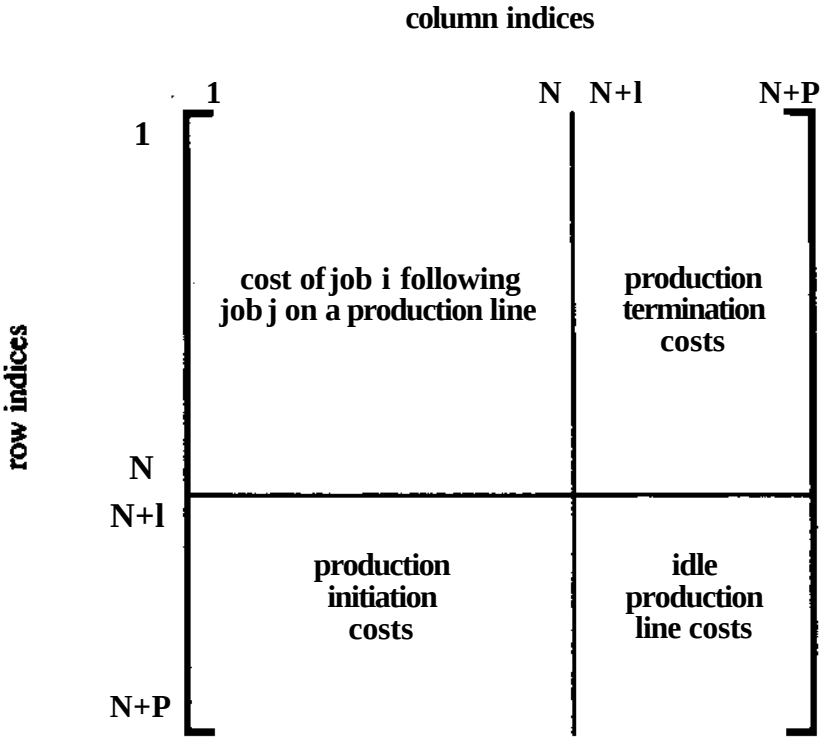


Figure 3 The structure of the TSP cost matrix for the no-wait flowshop scheduling problem

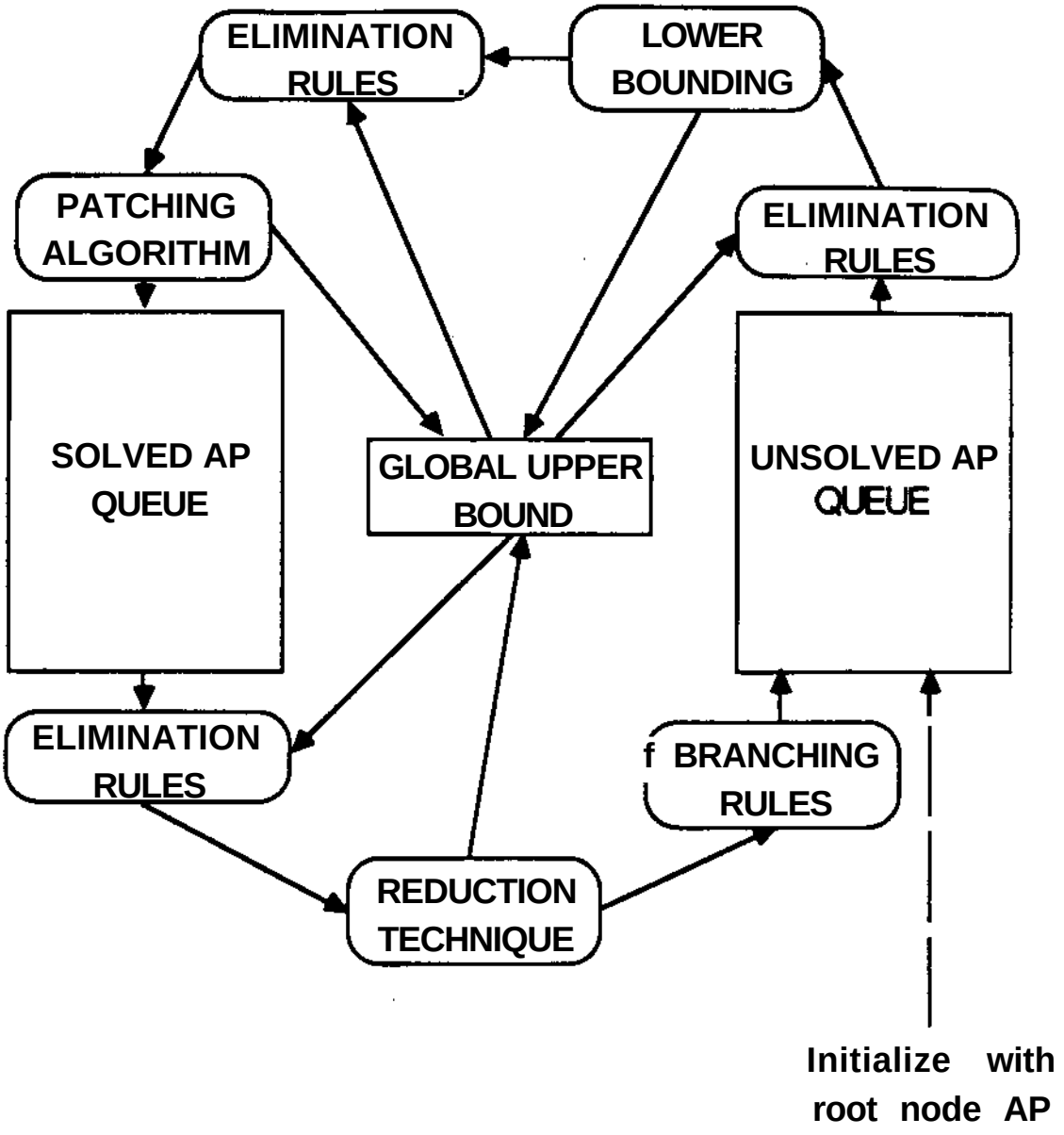


Figure 4 Data flow for parallel branch and bound algorithm

```
(1) if (unsolved AP queue not empty) then
        remove an AP; with the least lower bound from the unsolved queue,
        use elimination rules to delete APj if possible (if AP| eliminated, go to (1)).
        solve the assignment problem associated with APj.
        replace global upper bound if possible (if replaced, delete AP} and go to (1)).
        use elimination rules to delete AP\ if possible (if AP[ eliminated, go to (1)).
        apply patching algorithm to APj and replace global upper bound if possible,
        place AP| on the solved AP queue,
        goto(l).
    end if.
(2) if (solved AP queue not empty) then
        remove an APfc with the least lower bound from the solved queue.
        use elimination rules to delete AP^ if possible (if APfc eliminated, go to (1)).
        if (reduction technique is successful on AP^) then
            replace global upper bound if possible, delete AP^, and goto (1).
        end if.
        apply branching rules to AP^ to generate new unsolved APs.
        place the new APs on the unsolved AP queue,
        delete AP^.
        goto(l).
    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

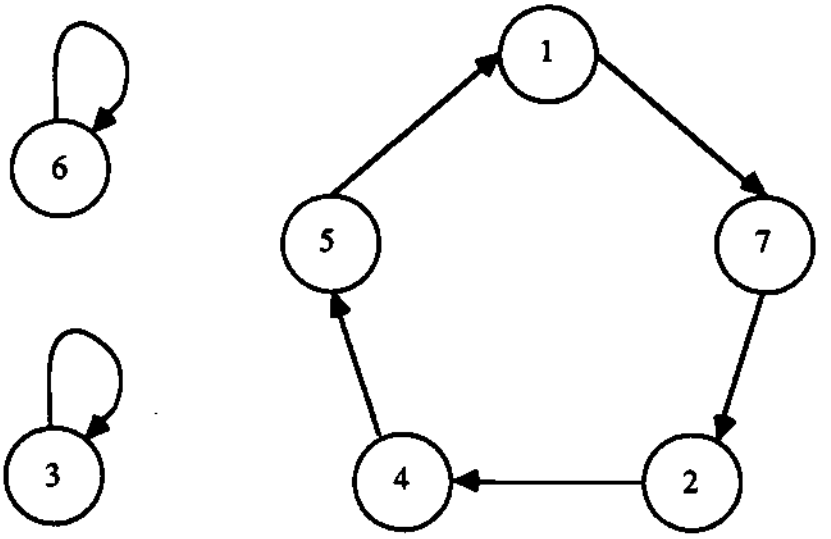


Figure 6 Graph representing a feasible TSP solution

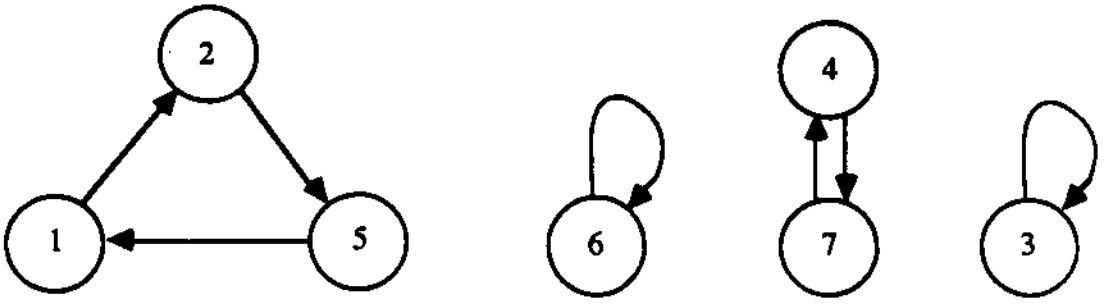


Figure 7 Graph representing an assignment problem solution that is an infeasible TSP solution

Table It

cost range [0,1000]

n	cases	time (avg/std) in sec.	AP solved (avg/std)	ratio (avg/std)
500	7	38.95/29.30	56.86/91.66	1.002/0.003
1500	7	163.80/100.62	4.29/ 7.85	1.000/0.001
2500	7	824.25/629.62	9.57/14.68	1.001/0.001
3500	7	607.25/507.40	1.00/ 0.00	1.000/0.000

cost range [0,10000]

n	cases	time (avg/std) in sec.	AP solved (avg/std)	ratio (avg/std)
500	7	146.15/99.92	558.57/585.62	1.002/0.002
1500	7	1071.69/555.50	478.14/305.57	1.000/0.000
2500	7	2669.28/1710.54	449.43/436.76	1.000/0.000
3500	7	1724.62/1001.97	57.57/62.74	1.000/0.000

t data collected on a 14 processor BBN Butterfly Plus

Table 2f

cost range [0,1000]

processors	cases	time (avg/std) in sec.	AP solved (avg/std)
1	7	64.93/49.09	28.00/30.04
2	7	44.23/26.98	27.29/27.16
4	7	48.83/47.71	47.57/65.68
8	7	32.09/16.52	30.43/29.74
14	7	38.95/29.30	56.86/91.66

cost range [0,10000]

processors	cases	time (avg/std) in sec.	AP solved (avg/std)
1	7	920.05/1168.72	228.43/181.86
2	7	606.84/1014.40	424.00/455.77
4	7	225.57/194.24	497.29/480.75
8	7	158.82/114.50	567.14/566.72
14	7	146.15/99.92	558.57/585.62

t 500 city problem data collected on a BBN Butterfly Plus

Table 3

cost range [0,1000]

n	without reduction technique! time (avg) in sec.	without reduction technique! AP solved (avg)	with reduction technique^ time (avg) in sec.	with reduction technique^ AP solved (avg)
500	54.52	138.9	38.95	56.86
1500	571.9	284.7	163.8	4.29
2500	1915.3	292.9	824.3	9.57

cost range [0,10000]

n	without reduction technique t time (avg) in sec.	without reduction technique t AP solved (avg)	with reduction technique}: time (avg) in sec.	with reduction technique}: AP solved (avg)
500	129.1	509.6	146.2	558.6
1500	1861.3	866.4	1071.7	478.1
2500	6277.8	1340.8	2669.3	449.4

t average of ten runs for each problem size using a 14 processor BBN Butterfly
t average of seven runs for each problem size using a 14 processor BBN Butterfly