Admissible Search Methods for Minimum Penalty Sequencing of Jobs with Setup Times on One and Two Machines

Anup K. Sen, A. Bagchi and Bani K. Sinha
Indian Institute of Management Calcutta
Joka, Diamond Harbour Road
P.O. Box 16757, Calcutta - 700 027, India

Abstract

Many efficient implementations of AI search algorithms have been realized in recent years. In an effort to widen the area of application of search methods to problems that arise in industry, this paper examines the role that search can play in solving certain types of hard optimization problems that involve the proper sequencing of jobs in one-machine job-shops and two-machine flow-shops. The problems studied here have the following general form: The completion of a job at time f induces a penalty G(f), where G(.) is a given penalty function which can be different for different jobs; the jobs must be so sequenced that the total penalty summed over all jobs is minimized. The objective is to improve upon current methods and to show that problems earlier considered formidable can at least be attempted, if not resolved satisfactorily, using admissible search algorithms. The crucial aspect is the derivation of good admissible heuristics that can direct the search narrowly to a goal. The search graph is not necessarily a tree. Algorithm A* has been run on randomly generated data using the derived heuristic estimates to solve a variety of penalty minimization problems; some indicative experimental results are provided.

1 Introduction

Recent years have witnessed a renewed spurt of interest in the theory and implementation of best-first search techniques [Pearl, 1984]. This interest is yet to be adequately reflected, however, in the application of AI search methods to optimization problems that arise in industry, many of which cannot be solved conveniently by any of the known algorithms. Viswanathan and Bagchi [1988] describe a successful attempt at applying search methods to rectangular cutting stock problems, but such examples are few. In this context, an area that appears to hold great promise is *sequencing and scheduling,* where most problems of practical importance require the use of branch-and-bound or dynamic programming methods [French, 1982; Lawler *et al.*, 1982]. Now that efficient implementations of best-first search algorithms such as A*, IDA* [Korf, 1985], and MREC [Sen and Bagchi, 1989] have been realized, search could prove superior to other methods in cases where reasonably tight lower bounds can be derived.

This paper investigates the role that best-first search can play in solving certain types of one-machine and two-machine job sequencing problems that arise in job-shops and flow-shops. The major objective is to enlarge the sphere of application of best-first search, and at the same time to illustrate how good admissible heuristics can be defined for practical problems. A typical problem studied by us has the following form. Jobs J,, $1 \leq i \leq n$, with processing times a_i $1 \leq i \leq n$, have been submitted to a one-machine job-shop at time t = 0. The jobs are to be processed on the given machine one at a time. Let the processing of job J_i be completed at time f_i. *Penalty functions* $G_i(\cdot)$, $1 \leq i \leq n$, are supplied, such that the penalty associated with completing job J_j at time f_i is $G_i(f_i)$. An example of a penalty function is $G_i(f_i) = c_i f_i^2$, where the c_i's are given constants, the penalty associated with a job being proportional to the square of the finish time of the job. The jobs must be sequenced on the machine in such a way that the total penalty

$$F = \sum \{ G_i(f_i) \mid 1 \leq i \leq n \}$$

is minimized. The penalty functions arc non-decreasing, and in general non-linear. Moreover, a job has a *setup time* that is independent of its position in the processing sequence; we say that the setup times are *separable.* In a more general formulation, the setup times can be *sequence-dependent,* i.e. the setup time of a job can depend on the job that immediately precedes it in the processing sequence. Problems of this type have obvious relevance to industry, but are known to be very hard to solve satisfactorily in their full generality, the number of job sequences being exponential in the number of jobs. No efficient general methods of solution have been reported. Schild and Fredman [1963] proposed a scheme for non-linear penalty functions that ignored setup times but nevertheless required a great deal of computation; improvements were subsequently suggested by Townsend [1978] and Bagga and Kalra [1980], but these were specific to quadratic penalty functions. Another special case has been studied by Rinnooy Kan *et al.* [1975]. As for two-machine flowshop situations, work has been confined to the determination of minimum length

schedules in the presence of setup times (see, for example, [Gupta and Darrow, 1985; Sule, 1982; Corwin and Esogbue, 1974]); penalty functions have not yet been considered by anyone in any detail.

This paper, which is in the nature of a preliminary report, describes how best-first search techniques can be used to determine minimal penalty job sequences for one and two machines when the setup times are separable. A* is the search algorithm used. Each case involves the computation of a reasonably tight lower bound that can serve to guide the search fairly narrowly to a goal. Search graphs have been implemented as graphs and not as trees whenever it has been found possible to do so; this tends to speed up the execution. In the two-machine case the search methods given below are not always *order preserving* [Pearl, 1984, p. 102], and unless the search graph is a tree the solution may not be optimal. The programs have been run on random data that try to simulate real-life situations. The results we have obtained are most encouraging; our main contribution lies in showing how useful best-first search can be in attacking job sequencing problems traditionally regarded as very difficult, such as the determination of a minimum penalty job sequence for a two-machine flow-shop however simple the penalty function, or for a single machine when the penalty for a job is proportional to a non-integral power of the finish time of the job.

2 One Machine Job Sequencing Problems

2.1 Problem Description

For the one-machine problem with sequence-dependent setup times we define the following notation :

n	number of jobs
J_i	ith job
a_i	processing time (> 0) for job J_i
s_{ij}	setup time for job J_j when it immediately follows job J_j in the processing sequence
f_i	completion time of job J_i
d_i	deadline for job J_i
$G_i(.)$	penalty function for job ji
c_i	penalty coefficient for job J_i
F	total penalty for a given sequence

If J_i is the first job in the sequence, then $s_{0,i}$ represents its setup time. After the last job in the sequence is completed, the machine need not be brought back to any specific state. A penalty of $G_i(f_i - d_i)$ is incurred for job J_i when $f_i > f_i$; if $f_i \leq d_i$ then no penalty is incurred for J_i. The formulation can be generalized to include the cost of carrying in inventory those jobs that get completed before their respective deadlines. In order to simplify the problem it is frequently assumed that all deadlines are zero; we make this assumption in the paper.

When the setup times are separable as in [Sule, 1982], we have the following alternative notation

p_i	initialization time for job J_i
q_i	cleanup time for job J_i
A_i	Pi + i + q_i

Initialization time refers to the time needed to initialize the machine settings before job J_i can be processed, while cleanup time refers to the time needed to clean the machine and bring it back to a specified state after the processing of job J_i has been completed. When job J_j follows job J_i in the sequence, the total setup time (q_i + P_j) can be viewed as corresponding to s_{ij} in the general case; again, if J_i is the first job in the sequence, then p_i corresponds to $s_{o,i}$ in the general case. Let the jobs be processed in the sequence

$$J_{j(l)}, J_{J(2)}, \ldots J_{j(n)}$$

and let the completion time of job $J_{j(k)}$ be $fj_{(k)}$. Then for $1 \leq k \leq n$, $f_{j(k)} = \sum \{ A_{j(i)} \mid 1 \leq i \leq k \} - q_{j(k)}$.

Note that the finish time of a job does not include the cleanup time for the job; the cleanup time gets added when the finish times of the jobs that follow in the sequence are computed. For one-machine problems, there is no loss of generality if the initialization time of a job is included in the processing time and not considered separately. As an illustration of the kind of problem we are trying to solve, consider the following example:

Example: Penalty for a job J_i proportional to its finish time f_i; separable setup times.

Here, $G_i(f_i) = c_i f_i$, where the ci's are constants, C_i being the penalty coefficient for job J_j. Thus

$$F = \sum \{ c_{j(k)} f_{j(k)} \mid 1 \leq k \leq n \}$$
$$= \sum_k c_{j(k)} \sum \{ A_{j(i)} \mid 1 \leq i \leq k \} - \sum_k c_k q_k$$

To minimize F we just need to arrange the jobs in the order [French, 1982]:

$$A_{j(1)}/c_{j(1)} \leq A_{j(2)}/c_{j(2)} \leq \cdots \leq A_{j(n)}/c_{j(n)},$$

since the last sum in the expression for F is a constant. Here we do not need to take help of search methods. □

In more complex situations we are forced to take recourse to AI search methods. Algorithm A* is implemented in the standard manner [Nilsson, 1980; Pearl, 1984]. OPEN is maintained as a priority queue. Nodes generally correspond to unordered subsets of jobs; the root node is the empty set. When a node corresponding to a subset of k jobs gets selected from OPEN and expanded, it generates (n-k) sons, each son being a subset of (k+1) jobs. A problem gets solved when a complete set of n jobs gets selected from OPEN. For two-machine problems, a node sometimes corresponds to an ordered sequence of jobs. The following additional notation is used in the specifications of the problems:

N	current node (either an unordered subset of jobs or an ordered partial sequence of jobs)
$g(N)$	cost of minimal cost path currently known from the root node to node N
$h(N)$	heuristic estimate at node N
$h^*(N)$	cost of minimal cost path from node N to a goal node
P	set of jobs already processed at node N
Q	set of jobs remaining to be processed at node N
T	time at which all jobs in P get completed
q_{last}	cleanup time of last job in P
m	number of jobs in Q

2.2 Non-Linear Penalty Functions

Case I Penalty for a job proportional to square of its completion time; all initialization and cleanup times are zero.

Here $G_i(f_i) = c_i f_i^2$. Townsend [1978] proposed a new method for computing lower bounds in a best-first branch-and-bound algorithm to solve this problem; certain improvements were subsequendy suggested by Bagga and Kalra [1982]. We compute the heuristic estimate h(N) at the current node N as follows :

Order the processing times of the m jobs in Q in the non-decreasing order

$$a_{i1)} \leq a_{j(2)} \leq \cdot \cdot * \leq a_{j(m)}$$

Also order the penalty coefficients of the jobs in Q *independently* in the non-increasing order

$$c_{i(I)} \geq c_{i(2)} \geq \ldots\ldots\ldots \geq c_{i(m)'}$$

Take the heuristic estimate as

$$h(N) = \sum \{ c_{i(k)} f_{j(k)}^2 \mid 1 \leq k \leq m \}$$

where $$f_{j(k)} = T + \sum \{ a_{j(u)} \mid 1 \leq u \leq k \},$$

and $$T = \sum \{ a_k \mid J_k \text{ is in } P \}.$$

Our computation ensures that h(N) is a lower bound on h*(N), the cost of the optimal path from N to a goal, since the penalty functions strictly dominate each other and have no crossover points. The sorting of the processing times and the penalty coefficients is done only once at start. Thereafter, we just have to keep track of which jobs are in Q, and when computing the heuristic estimate we simply skip over those terms in the sorted lists that correspond to jobs in P.

The method can be generalized to handle penalty functions for job J_i that are proportional to $(f_i)^k$ or to $\exp(kf_i)$, where k is a real constant independent of i. Such functional forms have not been considered by Townsend [1978] or Bagga and Kalra [1980].

In our experiments, programs were written in C and run on the SUN 3/60 workstation. The search graph was a graph and not a tree. The heuristic is *monotone* [Nilsson, 1980], so that a node is expanded at most once. But it is still advantageous to use a graph representation because this allows the g-values of unexpanded nodes to get updated in OPEN. A tree representation causes an explosion in the number of nodes generated and expanded. To implement graphs efficiently, we need to be able to find out quickly whether a newly generated successor is already present in the graph. This can be achieved by the use of a *hashing scheme.* A node N corresponds to a subset of jobs; the bit representation of N is viewed as a binary number which is then hashed using the modulo method. All particulars about a node are stored in the hash table; entries in the priority queue are pointers to the hash table. Two different penalty functions of the form $G_i(f_i) = c_i(f_i)^k$ were considered, with k = 2.0 and k = 1.5. Processing times (ai) of jobs were integers and were chosen randomly in the range 1 to 99 from a uniform distribution. Penalty coefficients (c_i) were also integers and were generated in the range 1 to 9 in the same way. For each set of jobs, the execution time, the total number of nodes generated, and the total number of nodes expanded were averaged over 25 runs. Results are given in Table I. It was found experimentally that for k = 2, Townsend's algorithm runs slightly faster than ours, but these results are not shown in the table.

TABLE I

	k = 2.0			k = 1.5		
Number of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp	Exec Time (sec)	Nodes Gen	Nodes Exp
12	0.83	2118	855	2.39	1799	669
13	1.63	3980	1478	4.63	3266	1092
14	3.29	7561	2871	9.36	6130	2098
15	7.43	14123	5519	19.28	11322	3953

For k = 1.5, computations were done using real numbers, and so the execution times are larger. The heuristic is quite tight: For 14 jobs, only 7561 nodes are generated when k = 2.0, while there are 14! job sequences.

Case II Penalty for a job proportional to square of its completion time; separable setup times.

No algorithm has yet been proposed for this problem. We determine the heuristic estimate h(N) as follows. Let

$$q = \min \{ q_k \mid J_k \text{ is in } Q \}$$

Order the jobs in Q according to the conditions

$$(p_{j(1)} + a_{j(1)}) \leq (p_{j(2)} + a_{j(2)}) \leq \ldots \leq (p_{j(m)} + a_{j(m)}),$$

and order the penalty coefficients as in Case I. The expression for h(N) is the same as in Case I, but here take

$$f_{j(k)} = T + q_{last} + \sum \{ (p_{j(u)} + a_{j(u)}) \mid 1 \leq u \leq k \} + (k - 1)q$$

and $$T = \sum \{ A_k \mid J_k \text{ is in } P \} - q_{last}.$$

As in Case I, $h(N) \leq h^*(N)$, and h is monotone. Again, the method can be extended to powers other than two and to exponentials. A* was run with the above heuristics. Data was generated as in Case I; the initialization and cleanup times were also generated randomly, but in the range 1 to 9 to keep them smaller than the processing times. Nodes still correspond to unordered subsets of jobs. Results are shown in Table II for the case k = 2. A depth-first branch-and-bound implementation was also tried out, but was found to be much slower in execution.

TABLE II

Number of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp
10	0.18	535	191
12	0.75	1910	676
14	2.93	6647	2246

Case III General non-linear penalty functions; separable setup times.

This generalizes Case II. Here we consider penalty functions G_i that have the non-negative integers as

domain and the non-negative real numbers as range; each G_i is non-decreasing and possibly non-linear, and the Gi's do not necessarily have identical functional forms [Schild and Fredman, 1963]. Let q, T and $f_{j(k)}$ be defined as in Case n, and let

$$G(x) = \min \{ G_i(x) \mid J_i \text{ is in } Q \}.$$

When computing the heuristic estimate arrange the jobs as in Case II and then use G and q instead of the G_i's and the qi's, but when computing g(N) use the actual G_i and q_i values. Close bounds are hard to obtain when the G_i's have different functional forms.

Claim $h(N) < h^*(N)$.

Proof: Express $h^*(N)$ as a function of T, and of the parameters G_i, p_i, a_i and q_i of the jobs J_i in Q. Replace each occurrence of G_i and q_i in the expression for $h^*(N)$ by G and q respectively; this gives a value $h_o N) < h^*(N)$. But when G and q are used for computing the heuristic estimate, h(N) is the smallest estimate we can get, so that we must have $h(N) < ho(N)$. □

Some experimental results are given in Table 1H. As before, nodes correspond to unordered sets of jobs and the heuristic is monotone. The penalty functions were chosen to have the form $G_i(f_i) = C_i f_i + c_i (f_i)^2$. The coefficients C_i and c_i were both randomly generated in the range 1 to 9.

Table III

Number of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp
10	1.51	1011	775
11	3.54	2019	1624
12	8.25	4091	3400

3 Two-Machine Flow-Shop Problems

In the two-machine flow-shop problem, each job must be processed twice; it must be processed on the first machine before it is processed on the second machine. The completion time of a job is the time at which its processing on the second machine is finished. The classic algorithm of Johnson [1954] finds a minimum length schedule. Our objective is to minimize the total penalty. This problem has been generally considered to be intractable, and there are very few references on the topic. Search methods can be applied here with advantage. We use the notation introduced earlier with the following additions and modifications:

M_1	machine 1
M_2	machine 2
p_i, a_i, q_i	initialization, processing and cleanup times of J_i on M_1
r_i, b_i, s_i	initialization, processing and cleanup times of J_i on M_2
A_i	$p_i + a_i + q_i$
B_i	$r_i + b_i + s_i$
T_1	completion time on M_1 of all jobs in P
T_2	completion time on M_2 of all jobs in P
q_{last}	cleanup time on M_1 of the last job in P
s_{last}	cleanup time on M_2 of the last job in P

A well known theorem (see French [1982], Theorem 5.1, p. 67) states that in the absence of setup times, there is an optimal schedule in which jobs are sequenced in the same order on the two machines. This theorem remains valid when the setup times are separable, but not when they are sequence-dependent [Gupta and Darrow, 1985]. Since our study is confined to separable setup times, we can restrict ourselves to job sequences which are identical for the two machines.

Case I Penalty for a job equal to its completion time;
all initialization and cleanup times are zero.

Here g(N) is the sum of the finish times of the jobs in P. To determine h(N), order the processing times on M_1 of the m jobs in Q as follows:

$$a_{i(1)} \le a_{i(2)} \le \ldots \le a_{i(m)}$$

Order the processing times on M_2 of the jobs in Q *independently* as follows:

$$b_{j(1)} \le b_{j(2)} \le \ldots \le b_{j(m)}$$

Let $X = mT_1 + \sum \{ (m + 1 - k)\, a_{i(k)} + b_k \mid 1 \le k \le m \}$

$Y = mT_3 + \sum \{ (m + 1 - k)\, b_{j(k)} \mid 1 \le k \le m \}$

where $T_3 = \max \{ T_1 + a_{i(1)}, T_2 \}$,

and take $h(N) = \max \{ X, Y \}$.

Claim $h(N) \le h^*(N)$.

Proof We have to show that $X \le h^*(N)$ *and* $Y \le h^*(N)$. The jobs in Q have been so ordered that the finish time on M_1 of the kth job in Q is the earliest possible; moreover, each of the jobs in Q must be processed on M_2, and the processing on M_2 of the kth job in Q cannot begin until the processing on M_1 of the kth job is over. This proves that $X \le h^*(N)$. Again, the processing of any job in Q cannot commence on M_2 before T_3, so $Y \le h^*(N)$. □

Sequencing problems of up to 12 jobs can be readily solved using a tree representation for the search graph, where a node corresponds to an ordered sequence of jobs. Such a formulation is essentially equivalent to a best-first branch and bound procedure. If we represent a node as an unordered subset of jobs we find that the search is faster but no longer order preserving, because when the jobs in P are permuted, not only g(N) but also T_2, and therefore T_3, get altered. This has the unusual consequence that with a graph representation, non-optimal solutions are outputted even when the heuristics are admissible. It was found experimentally, however, that the solutions are close to optimal, so that the method can still be useful in practice. Experimental results are given in Table IV. Data was generated

TABLE IV

	Tree				Graph			
No of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp	Opt Cost	Exec Time (sec)	Nodes Gen	Nodes Exp	Cost Obtd
10	0.29	1192	210	2652	0.28	268	68	2660
11	0.86	3343	563	3212	0.72	441	103	3226
12	4.59	15741	2622	3723	2.56	834	204	3734

independently and randomly for the jobs on the two machines. It is interesting to observe that if we change the expressions for X and Y to

$$X = T_1 + \sum \{ a_k \mid 1 \le k \le m \} + b_{j(1)}$$
$$Y = T_3 + \sum \{ b_k \mid 1 \le k \le m \}$$

and set g(N) = 0 and h(N) = max { X, Y }, we get a minimum length schedule. This gives us an alternative to Johnson's algorithm, but the latter runs much faster.

Case II Penalty for a job proportional to its completion time; separable setup times.

This generalizes Case I. Here $G_i(f_i) = c_i f_i$, where the c_i's are the penalty coefficients as before. Let g(N) be the weighted sum of the finish times of the jobs in P. To compute h(N) order the processing times on M_1 of the m jobs in Q as follows:

$$A_{i(1)}/c_{i(1)} \le A_{i(2)}/c_{i(2)} \le \ldots \le A_{i(m)}/c_{i(m)}$$

Independently order the processing times on M_2 of the jobs in Q:

$$B_{j(1)}/c_{j(1)} \le B_{j(2)}/c_{j(2)} \le \ldots \le B_{j(m)}/c_{j(m)}$$

Also, let D be the inner product, for the jobs in Q, of the m-element b-vector with the corresponding c-vector, where the b_i's in the b-vector are arranged in non-decreasing order while the c_i's in the c-vector are arranged in non-increasing order. Now take

$$X = \sum_{k=1}^{m} (T_1 + q_{last} + \sum \{ A_{i(u)} \mid 1 \le u \le k \}) c_{i(k)} + D - \sum \{ q_k c_k \mid J_k \text{ is in } Q \}$$

$$Y = \sum_{k=1}^{m} (T_3 + \sum \{ B_{j(u)} \mid 1 \le u \le k \}) c_{j(k)} - \sum \{ s_k c_k \mid J_k \text{ is in } Q \}$$

where $T_3 = \max \{ T_1 + q_{last} + p_{i(1)} + a_{i(1)} - r_{max}, T_2 + s_{last} \}$

$r_{max} = \max \{ r_k \mid J_k \text{ is in } Q \}$.

Let h(N) = max { X, Y }. Experimental results are given in Table V. Here, too, h(N) is a lower bound on h*(N). If we can redefine X and Y as follows

$$X = T_1 + q_{last} + \sum \{ A_k \mid 1 \le k \le m \} + b_{j(1)} - q_{max}$$
$$Y = T_3 + \sum \{ B_k \mid 1 \le k \le m \} - s_{max}$$

where $q_{max} = \max \{ q_k \mid J_k \text{ is in } Q \}$, and
$s_{max} = \max \{ s_k \mid J_k \text{ is in } Q \}$

and set g(N) = 0 and h(N) = max { X, Y }, we get an alternative to Sule's [1982] method for finding a minimum length schedule when setup times are separable.

TABLE V

Number of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp
10	0.34	1140	208
11	0.83	2670	439
12	4.15	12241	1926

Case III Penalty for a job proportional to square of its completion time; all initialization and cleanup times are zero.

The penalty functions are as in Case II of Section 2.2. Compute g(N) by summing, over the jobs in P, the squares of the finish times multiplied by the appropriate penalty coefficients. Order as in Case I the processing times on M_1 of the jobs in Q, and independently the processing times on M_2 of the jobs in Q. Also order the corresponding penalty coefficients c_i in non-increasing order:

$$c_{k(1)} \ge c_{k(2)} \ge \ldots \ge c_{k(m)}$$

Let $X = \sum_{u=1}^{m} c_{k(u)} [(T_1 + \sum \{a_{i(w)} \mid 1 \le w \le u\})^2 + (b_{j(u)})^2]$

$$Y = \sum_{u=1}^{m} c_{k(u)} (T_3 + \sum \{b_{j(w)} \mid 1 \le w \le u\})^2$$

$$Z = \sum_{u=1}^{m} c_{k(u)} (T_1 + \sum \{a_{i(w)} \mid 1 \le w \le u\} + b_{j(1)})^2,$$

where T_1 and T_3 are as in Case I, and take

h(N) = max { X, Y, Z }.

We again have $h(N) \le h^*(N)$. This case, as well as the next, can be generalized to powers other than two and to exponentials. Experimental results for the square penalty function are given in Table VI.

TABLE VI

Number of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp
8	0.57	1730	453
9	1.81	4965	1181
10	13.90	33750	7828

Case IV Penalty for a job proportional to square of its completion time; separable setup times.

In this case let $q = \min \{ q_k \mid J_k \text{ is in } Q \}$
$s = \min \{ s_k \mid J_k \text{ is in } Q \}$

Order the processing times on M_1 and M_2 of the jobs in Q independently as follows:

$$(p_{i(1)} + a_{i(1)}) \le (p_{i(2)} + a_{i(2)}) \le \ldots \le (p_{i(m)} + a_{i(m)})$$
$$(r_{j(1)} + b_{j(1)}) \le (r_{j(2)} + b_{j(2)}) \le \ldots \le (r_{j(m)} + b_{j(m)})$$

Let the penalty coefficients be arranged in non-increasing order as in Case III. Arrange the m-element b-vector in non-decreasing order, square each element, and let D be the inner product of the resulting vector with the corresponding m-element c-vector arranged as above.

TABLE VII

Number of Jobs	Exec Time (sec)	Nodes Gen	Nodes Exp
8	0.46	1084	260
9	3.25	6917	1632

Now let b_{Qmin} be the smallest b_i in Q, and let

$$X = \sum_{u=1}^{m} c_{k(u)} [T_1 + q_{last} + \sum \{(p_{i(w)} + a_{i(w)}) \mid 1 \le w \le u\} + (u-1)q]^2 + D$$

$$Y = \sum_{u=1}^{m} c_{k(u)} [T_3 + \sum \{(r_{j(w)} + b_{j(w)}) \mid 1 \le w \le u\} + (u-1)s]^2$$

$$Z = \sum_{u=1}^{m} c_{k(u)} [T_1 + q_{last} + \sum \{(q_{i(w)} + a_{i(w)}) \mid 1 \le w \le u\} + (u-1)q + b_{Qmin}]^2$$

and take h(N) = max { X, Y, Z }

Experimental results are given in Table VII.

4 Conclusion

The major aim of this paper has been to open a new area of application for AI search methods, and to illustrate how good admissible heuristics can be derived for some practical problems. It has been shown that certain types of difficult but practical one-machine and two-machine job sequencing problems, for which efficient algorithms are not currently known, can be tackled in a reasonably satisfactory way by admissible search schemes. The work reported here can be extended in many directions. Even before any extension in scope is attempted, more intensive experimental testing needs to be undertaken of the admissible heuristics derived here on real life data gathered from job-shops and flow-shops. The following additional issues and questions could be taken up for closer examination and study:

i) Is it possible to modify the methods described here to solve sequencing problems involving sequence-dependent setup times? In the one-machine case, this problem has some similarities with the travelling salesman problem. In the two-machine case, an optimal schedule can sequence jobs in different orders on the two machines; this greatly complicates the situation, and perhaps a totally new approach *is* necessary.

ii) When there are more than two machines in a flow-shop, an optimal schedule can order jobs in different orders on the machines. One way out of the difficulty is to restrict oneself to *permutation schedules* only [Lageweg *et* al., 1978], in which jobs are sequenced in the same order on all machines, but this has the obvious limitation that optimal schedules may not be realized. Can search methods be used to determine good permutation schedules ?

iii) In a real life problem, a job normally has a deadline by which the processing must be completed; no penalty accrues for a job that gets completed within its deadline. Is it possible to modify the heuristic estimate functions to take care of deadlines ? A positive answer will greatly increase the worth of this approach.

References

[Bagga and Kalra, 1980] P. C. Bagga and K. R. Kalra, A Node Elimination Procedure for Townsend's Algorithm for Solving the Single Machine Quadratic Penalty Function Scheduling Problem, *Management Science,* vol 26, no 6, 1980. pp 633-636.

[Cheng'en, 1989] Y. Cheng'en, A Dynamic Programming Algorithm for the Travelling Repairman Problem, *Asia-Pacific Journal of Operational Research,* vol 6, no 2, 1989, pp 192-206.

[Corwin and Esogbue, 1974] Corwin, B. D. and Esogbuc, A. 0., Two-Machine Row-Shop Scheduling Problems with Sequence Dependent Setup Times A Dynamic Programming Approach, *Naval Research Logistics Quarterly,* vol 21, 1974, pp 515-524.

[French, 1982] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop,* Ellis Horwood Ltd, 1982.

[Gupta and Darrow, 1985] J. N. D. Gupta and W. P. Darrow, Approximate Schedules for Two-Machine Flow-Shop With Sequence Dependent Setup Times, *Indian Journal of Management and Systems,* vol 1, no 1, 1985, pp 6-11.

[Johnson, 1954] S. M. Johnson, Optimal Two-and-Three-Stage Production Schedules With Setup Times Included, *Naval Research Logistics Quarterly,* vol I, 1954, pp 61-68.

[Korf, 1985] R. E. Korf, Depth-First Iterative Deepening. An Optimal Admissible Search, *Artificial Intelligence,* vol 27, no 1, 1985, pp 97-109.

[Lageweg *et al.,* 1978] B. J. Lageweg, J. K. Lenstra and A. H. G. Rinnooy Kan, A General Bounding Scheme for the Permutation Flow-Shop Problem, *Operations Research,* vol 26, no 1, 1978, pp 53-67.

[Lawler *et al.,* 1982] E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Recent Developments in Deterministic Sequencing and Scheduling: A Survey, in *Deterministic and Stochastic Scheduling* (Eds. M. A. H. Dempster, J. K. Lenstra and A. H. G. Rinnooy Kan), D Reidel Publishing Co., 1982.

[Nilsson, 1980] N. J. Nilsson, *Principles of Artificial Intelligence,* Tioga-Springer Verlag, 1980.

[Pearl, 1984] J. Pearl, *Heuristics,* Addison-Wesley, 1984.

[Rinnooy Kan *et al,* 1975] A. H. G. Rinnooy Kan, B. J. Lageweg and J. K. Lenstra, Minimizing Total Costs in One Machine Scheduling, *Operations Research,* vol 23, no 5, 1975, pp 908-927.

[Schild and Fredman. 1963] A. Schild and L J. Fredman, Scheduling Tasks with Deadlines and Non-linear Loss Functions, *Management Science,* vol 9, 1963, pp 73-81.

[Sen and Bagchi, 1989] Anup K. Sen and A. Bagchi, Fast Recursive Formulations for Best-First Search That Allow Controlled Use of Memory, *Proc. IJCA1-89,* International Joint Conference on Artificial Intelligence, Detroit, U.S.A., 1989, pp 297-302.

[Sule, 1982] Sule, D. R., Sequencing n Jobs on Two Machines with Setup, Processing and Removal Times, *Naval Research Logistics Quarterly,* vol 29, no 3, 1982, pp 517-519.

[Townsend, 1978] W. Townsend, The Single Machine Problem With Quadratic Penalty Function of Completion Times : A Branch and Bound Solution, *Management Science,* vol 24, no 5, pp 530-534.

[Viswanathan and Bagchi, 1988] K. V. Viswanathan and A. Bagchi, An Exact Best-First Search Procedure for the Constrained Rectangular Guillotine Knapsack Problem, *Proc. AAAI-88,* American Association for Artificial Intelligence, St. Paul, U.S.A., 1988, pp 145-149.