Exact and Local Search Methods For Three Machine Flow Shop with Transportation Times

Tariq.S.Abdul-Razaq and Hussein.J. Mutashar
Department of Mathematics ,College of Science, University of Mustansiriyah
Department of Mathematics ,College of Education, University of Misan

الخلاصة

تناولنا في هذا البحث مسألة جدولة (n) من النتاجات على ثلاث مكائن بوجود زمن للنقل بين تلك المكائن لتقليل اكبر وقت إتمام. نظرياً، تمكنا من اشتقاق اثنان من النتائج للأمثلية كحالات خاصة للمسألة. هذه المسألة معروفة على أنها من النوع المعقد (NP-hard) وأقترحنا خوارزميات للتفرع والتقيّد مع عدد من القيود الدُنيا المقترحة في هذا البحث والتي حصلنا عليها بعد ارخاء شرط (سعة الماكنة). بعد اختبار هذه القيود على مجموعة من المسائل المولدة عشوائياً. إن استخدام خوارزميات الأمثلية تبدو بأنها غير ضامنة ولذلك قمنا بمعالجة المسألة بطرائق البحث المحلية. كذلك قمنا بتطوير ومقارنة واختبار مختلف طرائق البحث المحلية:
(Descent Method, Simulation Annealing, Threshold Accepting, Tabu Search, Genetic Algorithm, Ant colony algorithm).
عملياً ومن خلال الخبرة الحسابية وجد، بأن خوارزميات البحث المحلي تستطيع حل المسألة إلى (7000) نتاج بوقت معقول.

ABSTRACT

This paper considers the problem of scheduling 'n' jobs on three machine flow shop with transportation times between the machines to minimize the maximum completion time. This problem is known as NP-hard. Theoretically, results concerning optimality for two special cases of the problem are given. Special attention is also given to branch and bound (BAB) and local search methods. The BAB algorithms use the quickly computed but possibly rather weak lower bounds obtained from relaxation of machines capacity constraints. The (BAB) algorithms are then tested on a large set of test problems.

Also, we develop, compare and test different local search methods (Descent, Simulation Annealing, Threshold Accepting, Tabu Search, Genetic Algorithm, Ant Colony Algorithm) for the problem. Computational experience is found that these local search algorithms solve the problem to 7000 jobs with reasonable time.

INTRODUCTION

A flow shop schedule problem can be stated as follows: there is a set of (M) different machines these machines perform tasks of (n) jobs, each of (n) jobs is processed by (M) machines (M_1, $M_{2, ...,}$ M_m) in this order. There are some constraints on jobs and machines, each machine can handle one job at a time and each job can be performed by one machine at a time. The scheduling of manufacturing systems has been the subject of extensive research since the early 1950s'. Many of the applications in the operational research field involve this type of problems. One of the first developments in flow shop scheduling problem is Johnson's algorithm (1), which shows the problem of scheduling (n) jobs on two machines for minimizing the completion time is solved by a single rule (Johnson's Rule "J.R."). Conway et. al.(2), observed that: for the problems of scheduling (n) jobs on (m) machines there exist an optimal schedule with the same processing order on the first two machines i.e. machine (1 and 2), and the same processing order on the last two machines i.e. machine (m-1 and m). The no-wait flow shop problem was discussed by Piehler (1960) (3), Bonney and Gundry (1976) (4), King and Spachis(1980)(5), and Rajendran (1993) (6). In this problem jobs are held before machine 1 and launched only when they can be sequentially processed by all m machines with out

delays at any of the machines. Johann Hurink (1998) (7), discuses ($F_2/L_i/C_{max}$) problem, and found some solvable special cases, in this problem the jobs are scheduling on two machine with transportation time between the machines.

1.1 The aim of this paper

This study has two tasks: the first is to improve the (BAB) algorithm for the problem, this improvement is presented by introducing new two lower bounds. The second goal is to use the local search methods for solving the problem with large size (n), where "n" is the number of jobs.

2. $F_3/l_i, k_i/ C_{max}$ problem

This section describes the problem considered in our work, this problem is a particular case of hole permutation flow shop scheduling problems with transportation time, in which we follow the commonly used three-field notation ($\alpha/\beta/\gamma$) for machine scheduling problems. In the (α) field, will be used notation 'F_3' to denote a three machines flow shop problem, we use A, B and C instead of M_1, M_2 and M_3 respectively. In β field we use "l_i" and "k_i" to denote, the transportation times respectively, from machine A to machine B and then from machine B to machine C. We use the notations a_i, b_i and c_i to denote, (respectively), the processing time of job i on A,B and machine C.
In γ field, we use the objective function "C_{max}" the total elapsed (completion) time "makespan". In this problem each job i process on machine A,B, and C in this order by processing time a_i, b_i and c_i, on machine A,B, and C, respectively. Hence, $F_3/l_i, k_i/ C_{max}$ represents the 3-machine flow shop makespan problem with (n) transporters.

3. Johnson's Rule for $F_2//C_{max}$ problem (J.R.)

An optimal sequence for $F_2//C_{max}$ problem was given by Johnson (1). It is determined by the following theorem:

Theorem (1): (8)

Suppose the set N of n jobs may be partitioned in to the following two sub-sets:
$N^a = \{J \in N \mid a_J \leq b_J\}$
$N^b = \{J \in N \mid a_J > b_J\}$,
where $a_J = P_{1J}$ and $b_J = P_{2J}$ $\forall J$.
For a schedule in which all jobs in N^a precede each of those in N^b, then the Job in N^a are sequence in non-decreasing order of a_J, while the Jobs in N^b are sequence in non-increasing order of b_J, the resulting sequence is an optimal with minimum C_{max} .□

3.1 Extension of Johnson's Rule (J.R)

The two machine flow shop case is easy (1), similarly the case of three machines is polynomially solvable under very restrictive requirements on processing times of the intermediate machine (9). We can obtain a criterion by which an optimal sequence, that is, limited solution can be determined under the constraint that no processing time on the first machine M_1 is smaller than on second machine M_2 or no processing time on the third

machine (last machine) M_3 is smaller than on second machine M_2. This fact is stated in the next theorem .

Theorem (2) : (8)

In the three machines flow shop min-makespan problem, let the ordering be M_1, M_2, M_3 then if:

a) $\min_i\{p_{i1}\} \geq \max_i\{p_{i2}\}$ or

b) $\min_i\{p_{i3}\} \geq \max_i\{p_{i2}\}$

hold , the optimal sequence can be determined by the next rule : if

$$\text{Min}\{P_{i1}+ P_{i2}, P_{j2}+ P_{j3}\} \leq \text{Min}\{P_{j1}+ P_{j2}, P_{i2}+ P_{i3}\}$$

holds without equality, job (i) precedes job (j), otherwise either ordering is optimal.□

This theorem means applied (J.R.) on the two artificial machine (α) and (β) with processing time ($P_{i1}+ P_{i2}$) and ($P_{i2}+ P_{i3}$) respectively, then calculate C_{max} of the sequence, which we found on the original (data) problem. The above theorem also can be extend for problem with (m) machines, extended and enlarging upon (J.R) general conditions for optimality have been obtained for general (m) machine flow shop (10), this extension can be describe as follows:

In m machine flow shop min-makespan problem. Let the ordering be $M_1 \ldots M_n$ then if :

a. $\min_i\{p_{i1}\} \geq \max_{i,J}\{p_{iJ}\}$ or

b. $\min_i\{p_{im}\} \geq \max_{i,J}\{p_{iJ}\}$ $\quad J = 2,\ldots,m-1 \quad i = 1,\ldots,n$

hold the optimal sequence (δ) which can be determined by applying (J.R.) on the two artificial machines (α) and (β) with processing times $\left(\sum_{k=1}^{m-1} p_{ik}\right) and \left(\sum_{k=2}^{m} p_{ik}\right)$ respectively, then calculate C_{max} of (δ) on the original data problem.

4. Solutions of machine scheduling problem

Combinatorial optimization problems and aso scheduling problems are concerned with the maximization or minimization of the value of an objective function, they consist of finding from among a finite set of alternatives one that optimizes the values of the objective function, and most of optimization problems are NP-hard (11).

The best-known methods of solution for machine schedule problems are generally divided into two types

The first one leads to an optimal solutions and is called "Exact Methods" which involve :

- Complete enumeration method.
- Dynamic programming (DP) method.
- Branch and Bound (BAB) method.

The BAB method will be used first, for solving $F_3/l_i, k_i/ C_{max}$ problem. We must noted here that in our BAB method, we start from the schedule (renumber the job-numbers) according to Johnson's rule (J.R.) on the two artificial machine (α) and (β) with processing times ($a_i+l_i+b_i+k_i$), ($l_i+b_i+k_i+c_i$) respectively.

The second one, that methods, which lead to near optimal solutions, it is called "Approximated methods" or "Heuristic Methods" which involve:

- Neighborhood search methods
    a) Descent Method (DM).
    b) Simulation Annealing (SA).
    c) Threshold Accepting (TA).
    d) Tabu Search (TS).
- Genetic Algorithm (GA).
- Ant Colony Optimization (ACO) Algorithm.

These methods will be described and used for solving ($F_3/l_i, k_i/ C_{max}$) problem.

Some analytical results for permutation three machines with transportation times flow shop scheduling problem are presented in the following sections (5, 6, 7).

5. Solvable special cases

The meaning of solvable special case for a machine schedule problem is that: if the data of this problem satisfies some conditions, then there is an optimal schedule for this problem, which can be found directly without using the exact methods such as (BAB, DP and complete enumeration method). Some special cases that are solvable in polynomially bounded computational effort can be identified.

Solvable special case 1

Let we have **($F_3 /l_i, k_i/ C_{max}$)** problem and one or both of the following condition is hold

I) $\min \{a_i\} \geq \max \{l_i, b_i, k_i\}$, $i = 1,\ldots,n$

II) $\min \{c_i\} \geq \max \{l_i, b_i, k_i\}$ $i = 1,\ldots,n$

then we can find an optimal sequence (δ) for this problem by applying (J.R.) on the two artificial machine (α) and (β) with processing times:

$\alpha_i = a_i+l_i+b_i+k_i$ $i = 1,\ldots,n$

$\beta_i = l_i+b_i+k_i+c_i$ $i = 1,\ldots,n$

respectively. Then calculate makespan of this sequence (δ) for the original problem, which is equal to the optimal value (minimum completion time on the last machine).

Indeed these conditions (I) and (II) come from when we assumed the transportation times (l_i) and (k_i) as a processing times for (L) and (K) machines. (Clearly, these machines are always ready to perform the jobs i.e. without any waiting times). Then our problem becomes as ($F_5 //C_{max}$) and then by using the extension of theorem (2).

Solvable special case 2

We can transfer the **($F_3/l_i, k_i /C_{max}$)** into (F_3 // C_{max}), by letting ($a'_i = a_i+l_i$, $b'_i=b_i+k_i$, $c'_i=c_i+l_i$) as a processing times for the three artificial machines A', B' and C' respectively. Now by using the theorem (2) we have: if one or both of the following conditions are hold:

I) $\min \{a'_i\} \geq \max \{ b'_i\}$

i.e. $\min \{a_i + l_i \} \geq \max \{ b_i + k_i\}$ (in origin)

II) $\min \{c'_i\} \geq \max \{ b'_i\}$

i.e. $\min \{c_i + l_i \} \geq \max \{ b_i + k_i\}$ (in origin)

then we can find an optimal sequence (δ) for this problem by applying (J.R.) on the two artificial machine (α) and (β) with processing times:

$\alpha_i = a_i+l_i+b_i+k_i$ $\quad i=1,\ldots..,n$
$\beta_i = l_i+b_i+k_i+c_i$ $\quad i=1,\ldots..,n$

respectively, then calculate makespan C_{max} (δ) of this sequence (δ) for the original problem.

6 Simple Heuristic algorithms

In this section, we propose heuristics methods, the best of them is applied at the root node of the BAB tree to find an upper bound (UB) on the minimum value of the problem $\mathbf{(F_3/l_i,k_i/C_{max})}$, which can be transfer it into $\mathbf{(F_3// C_{max})}$ problem.

For the first heuristic H1, let ($a'_i = a_i$, $b'_i = b_i+l_i$, $c'_i = c_i+k_i$)
For the second heuristic H2, let ($a'_i = a_i+l_i$, $b'_i = b_i$, $c'_i = c_i+k_i$)
For the third heuristic H3, let ($a'_i = a_i+l_i$, $b'_i = b_i+k_i$, $c'_i = c_i$),
where (a'_i, b'_i, and c'_i) as a processing times for the three artificial machines A', B' and C' respectively. Now by using the theorem (2), even when its conditions are not satisfied, Then we can find a sequence (δ), by applying (J.R.) on the two artificial machine (α) and (β) with processing times for each heuristic:

H1:
$\alpha_i = a'_i + b'_i = a_i + l_i + b_i$ $\quad i=1,...,n$
$\beta_i = c'_i + b'_i = l_i + b_i + k_i + c_i$ $\quad i=1,...,n$
H2:
$\alpha_i = a'_i + b'_i = a_i + l_i + b_i$ $\quad i=1,\ldots..,n$
$\beta_i = c'_i + b'_i = b_i + k_i + c_i$ $\quad i=1,\ldots..,n$
H3:
$\alpha_i = a'_i + b'_i = a_i + l_i + b_i + k_i$ $\quad i=1,\ldots..,n$
$\beta_i = c'_i + b'_i = b_i + k_i + c_i$ $\quad i=1,\ldots..,n$

respectively, then calculate makespan C_{max} (δ) of this sequence (δ) for the original problem. Also, we can use the heuristic (H4), which is given in [12] . For this heuristic H4 a sequence (δ) is obtained, by applying (J.R.) on the two artificial machine (α) and (β) with processing times :

$\alpha_i = a_i + l_i + b_i + k_i$ $\quad i=1,...,n$
$\beta_i = l_i + b_i + k_i + c_i$ $\quad i=1,...,n$

respectively, then calculate makespan C_{max} (δ) of this sequence (δ) for the original problem.

It is clear that heuristic H4 occasionally catch the optimal value if one of the conditions that had been given in solvable special cases sections is met.

In the following table, we list 10 examples with different number of jobs each one is solved by all heuristics:

N : number of jobs
EX : number of example
Hi = the value of heuristic i when i=1,2,3,4

$$hi = \begin{cases} 1 & \text{if Hi equal to the minimum value of current example} \\ 0 & \text{otherwise} \end{cases}$$

where i = 1,2,3,4

Table -1: Comparison of heuristic methods

N	EX	H1	H2	H3	H4	h 1	h2	h3	h4
5	1	682	623	706	616	0	0	0	1
10	2	972	1020	1005	906	0	0	0	1
13	3	1609	1638	1686	1666	1	0	0	0
15	4	2011	1943	1943	2308	0	1	1	0
20	5	3748	3755	3799	3568	0	0	0	1
25	6	5125	5170	5116	4847	0	0	0	1
30	7	5859	5807	6011	5680	0	0	0	1
40	8	7684	7611	7691	7690	0	1	0	0
50	9	9018	8756	9007	8917	0	1	0	0
60	10	10089	9761	9656	9873	0	0	1	0
the sum of (h1, h2, h3, h4 respectively)						1	3	2	5

From the above table (1) we conclude that: heuristic 4 (H4) is the beast one of the four heuristics.

7. Lower Bounds for (F_3 / l_i,k_i / C_{max}) problem

7.1. Single Machine Bound

Let **(F_3 / l_i, k_i / C_{max})** problem with n jobs to process on machines A,B, and C respectively, and let (δ) denote the sequenced jobs and (S) be set of unsequence jobs. This lower bound is determined by considering successive processing times and transportation times of unsequence jobs on the machines A,B, and C respectively, it is expressed as below :

Let $C^A(\delta)$, $C^B(\delta)$, and $C^C(\delta)$ be the completion times of δ on machine A,B and C respectively.

Now consider the following relaxation:

Machine A : If we release the constraint that machines B and C can process only one job at a time

$$LB1 = C^A(\delta) + \sum_{i \in s} a_i + \min_{i \in s}(l_i + b_i + k_i + c_i) \quad .$$

Machine B : If we ignore the processing times on machine A and the transportation time l_i, and we relax the constraint on machine C can process one job at a time

$$LB\,2 = C^{B}(\delta) + \sum_{i \in s} b_i + \min_{i \in s}(k_i + c_i)\,.$$

Machine C: If we ignore processing times on machines A, B and the transportation times

$$LB\,3 = C^{C}(\delta) + \sum_{i \in s} c_i \quad .$$

Hence our first lower bound
LB = max { LB1 , LB2 , LB3 }
We show next that solution of the relaxed problem provides a lower bound on the solution of the original problem.

Theorem (3) :

LB = max {LB1, LB2, LB3} is a lower bound for **($F_3/l_i,k_i/C_{max}$)** problem.

proof :

Let δ the sequenced jobs and let S be the set of unsequenced jobs. We shall prove that LB1, LB2 and LB3 are lower bounds.

For LB1

Let LB1(δ) be the finishing time of the processing on machine C, when initially all jobs in (S) are processed continuously without idle time on machine (A), after the time $C^A(\delta)$. Then find a job (i_*) in (S) which have minimum sum of ($l_{i*} + b_{i*} + k_{i*} + c_{i*}$) that job (i_*) is processed on machines (B) and (C) without waiting time. Then for the original problem, since any sequence of all jobs in (S) may produce idle time on machines (B and C), after the time $C^A(\delta)$, also the last job (i) in this sequence may have a sum of ($l_i+b_i+k_i+c_i$) which is not smaller than the sum of ($l_{i*} + b_{i*}+ k_{i*} +c_{i*}$). Hence, LB1 is a lower bound.

For LB2

Let LB2(δ) be the finishing time of the processing on machine C, when initially all jobs in (S) are processed continuously without idle time on machine (B), after the time $C^B(\delta)$. Then find a job (i_*) in (S) which have minimum sum of ($k_{i*} + c_{i*}$) that job (i_*) is processed on machine (C) without waiting time. Then for the original problem, since any sequence of all jobs in (S) may produce idle time on machines (B and C), after the time $C^B(\delta)$, also the last job (i) in this sequence may have a sum of (k_i+c_i) which is not smaller than the sum of ($k_{i*}+c_{i*}$), hence LB2 is a lower bound .

For LB3

Let LB3(δ) which is define as finishing time of the processing on machine C, when all jobs in (S) are processed continuously on machine (C), after the time $C^C(\delta)$, LB3 is a lower bound since usually the processing of jobs in (S) makes idle times on machines (B and C) .

Since LB1, LB2, LB3 are lower bounds then:
LB = max {LB1, LB2, LB3} is a Lower Bound for **($F_3/l_i,k_i/C_{max}$)** problem . □

We will denote this lower bound by (LB-I).

7.2. Lower Bound -II-

This lower bound is constructed by relaxation of the machine (B), i.e. we let the machine (B) always ready to perform the unsequence jobs without any waiting time. In this case the machine (B) becomes as transporter whose transportation time equal to the processing time of machine (B). Before starting this lower bound-II- some notes are required here: let (δ) be the sequenced jobs, and (S) be the set of unsequence jobs. In such a way that the new transportation time (t_i) will be the sum of ($l_i+b_i+k_i$), i.e. sum of the original transportation times (l_i ,k_i) and modified transportation time (b_i), $t_i=l_i+b_i+k_i$, for each job (i)∈(S). Our lower bound is found by order the jobs in (S) according to (J.R.) on the two artificial machine (α) and (β) with processing time ($\alpha_i=a_i+l_i+b_i+k_i$) and ($\beta_i=l_i+b_i+k_i+c_i$) respectively. For the unsequence jobs (S) of relaxed problem, the completion times (C^{*A}) and (C^{*C}) are computed as follows :

Let (r) be the last job in (δ) and (r +1) be the first job in (S) to be schedule

$C^{*A}(1) = C^{A}(r) + a_{r+1}$

$T'(1) = C^{*A}(1) + t_{r+1}$

$C^{*C}(1) = T'(1) + c_{r+1}$

Hence for any job $i \in S$, i= 2 ... k, where (k) is the last job of (S)

$C^{*A}(i) = C^{*A}(i-1) + a_i$

$T'(i) = C^{*A}(i) + t_i$

$C^{*C}(i) = \max \{ C^{*C}(i-1), T'(i)\} + c_i$

Where

$C^{*A}(1)$ is the completion time on machine A of job (1) in S

$C^{*C}(1)$ is the completion time on machine C of job (1) in S

$C^{*A}(i)$ is the completion time on machine A of job (i) in S

$C^{*C}(i)$ is the completion time on machine C of job (i) in S

Assume that :

$LB'1 = C^{*C}(k)$ (k) is the last job of (S)

$LB'2 = C^{C}(\delta) + \sum_{i \in s} c_i$

$LB = \max \{LB'1, LB'2\}$,

Next, the solution of the relaxed problem, which provides a lower bound on the solution of the original problem ($F_3/l_i,k_i/C_{max}$) is shown.

Theorem (4):

LB = max {LB'1, LB'2} is a lower bound for ($F_3/l_i,k_i/C_{max}$) problem.

Proof :

Let δ be the sequenced jobs and let S be the set of unsequence jobs

For LB'1

Let LB'1(δ) be the completion time of the processing on machine C. When all jobs in (S) are ordered by (J.R) on the two artificial machine (α) and (β) with processing time ($\alpha_i=a_i + l_i + b_i+ k_i$) and ($\beta_i = l_i + b_i + k_i + c_i$) respectively, and processed without waiting time on machine (B), since the machine (B) is relaxed. Hence any sequence of all jobs in (S) may

produce waiting time on machines (B and C) (for the original problem). These waiting times (greater than or equal to zero) are ignored in LB'1. Hence, LB'1 less than or equal to the minimum value of C_{max}.
i.e. LB'1 $\leq C_{max}$; then LB'1 is a lower bound .

For LB'2
Let LB'2(δ) the completion time of the processing on machine C. When all jobs in (S) are ordered by (J.R) on the two artificial machine (α) and (β) with processing time ($\alpha_i=a_i + l_i + b_i+ k_i$) and ($\beta_i = l_i + b_i + k_i + c_i$) respectively, and processed without idle time on machine (C), after the time $C^C(\delta)$. Since any sequence of all jobs in (S) may makes an idle time on machine (C). This idle time (greater than or equal to zero) is ignored in LB'2. Hence LB'2 less than or equal do the minimum value of C_{max}, i.e. LB'2 $\leq C_{max}$; then LB'2 is a lower bound.
Since LB'1and LB'2 are lower bounds then:
LB = max {LB'1, LB'2} is a Lower Bound for **($F_3/l_i,k_i/C_{max}$)** problem. □
We will refer to this lower bound by (LB-II).

8. Using (BAB) algorithm to solve ($F_3/l_i, k_i/C_{max}$) problem

We now give the main feature of our branch and bound algorithms, which are used for solving the (F_3/li, ki/ C_{max}) problem. The algorithms start by reorder (renumbered) the jobs according to (J.R.) on the two artificial machines (α) and (β) with processing times ($\alpha_i=a_i+l_i+b_i+k_i,\beta_i=l_i+b_i+k_i+c_i$) respectively. Prior to their application at the root node of search tree to obtain an upper bound (UB) by using the heuristics methods (H1,H2,H3,H4), introduced in section (6), and put UB=min{ H1,H2,H3,H4}. Also, at the root node of the search tree an initial lower bound (ILB) on the cost of an optimal schedule is obtained from {LB-I, LB-II}. The active new search procedure for flow shop scheduling problem is used to select a node, from which to branch.

8.1 Computational Experience with branch and bound algorithm

The branch and bund algorithms, were tested on ($F_3/l_i,k_i/C_{max}$) problem, with {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000} jobs. Number of jobs refers to the problem size. Job i become available for processing at a time zero, requires integers processing times (a_i, b_i, and c_i), i=1…n, were generated from the uniform distribution [1,100], and requires transportation times (l_i, k_i) i=1 …n , were generated from the uniform distribution [1,R], for R selected from {100, 250, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000}. For each value of (n) we have one problem for each value of (R). This yields (10) test problems, for each value of (n). This data is generated by Matlab language. We are coded (BAB) algorithms in Matlab language.

8.2 Comparison of the values of lower bounds(LB-I, LB-II)

Results comparing the two lower bounds. To make a comparison of the introduced lower bounds (LB-I, LB-II), we compute the optimal value, upper bound (UB), the lower bound (LB), the number of generated nodes (nodes), the computational time (time), and the

number of unsolved problems. To determined the number of unsolved problems, there are two criteria for stopping the (BAB) algorithm and say that this problem is unsolved or (this is big example). The first criteria is the number of generated nodes (i.e. terminate BAB algorithm after a fixed number of generated nodes). The second criteria which as follows: the (BAB) algorithm is stopped after a fixed period of time. For our comparison we chose the first criteria and our procedure will be stopped after generating (20,000,000) nodes. We solve (10) examples for each value of (n), where $n \in \{5,7,10,13,15\}$, in the following table we summarized the empirical results, where

n=number of jobs (problem size)

Av. nodes= average nodes for 10 examples of BAB algorithm with the assignment lower bound.

Av. time= average time by seconds for 10 examples of BAB algorithm with the assignment lower bound.

No. unsolved= the number of unsolved examples from 10 examples of BAB algorithm with the assignment lower bound.

Table -2:Comparison of branch and bound algorithms

n	Av.e nodes		Av. time		No. unsolved	
	LB-I	LB-II	LB-I	LB-II	LB-I	LB-II
5	85.5	36.3	0.03	0.03	0	0
7	1691.1	903	0.07	0.0431	0	0
10	510610.5	45155.8	15.69	0.9468	0	0
13	13198127	5757745	354.93	89.8684	6	2
15	17477547	12028590	460.48	197.7437	8	6

The above results of table(2) show that:

The mentioned results indicate the weakness of the lower bounds. The large number of unsolved problems for fairly small "n" (n=15) shows that introducing transportations times into a flow shop greatly increases the problem difficulty. An analysis of unsolved problems indicates that those with small "R" are relatively easy whereas those with large "R" are the hardest. The algorithm that uses lower bound-II- (LB-II), is the best one between the introduced algorithms since its computational time is the minimum of the other, and the number of generated nodes is the minimal one. Now for the algorithm that uses the lower bound (LB-I), the number of generated nodes for this algorithm is very large as compared to other algorithm, but the arithmetic process for its lower bound (LB-I) is very simple which makes the computational time associated with much smaller, in spite of above mentioned fact. The lower bound-I- (LB-I), is weaker than (LB-II), since the algorithm that uses the lower bound-I- (LB-I) which cannot solve some problems that are solved by the algorithm that use the lower bound (LB-II).

Hence, in the following sections if we refer to (BAB) algorithm we mean to the (BAB) algorithm that use the lower bound-II- (LB-II).

9. Local Search Heuristic Methods

It is clear to solve scheduling problems one tends to use (BAB) or (DP) to find optimal solutions, however, these approaches has two main disadvantages:

- It is mathematically complex and thus a lot of time to be invested
- When it concerns NP-hard problem, the computational time requirements are enormous for large sized problem.

To avoid these drawbacks we can appeal to heuristics methods. In the recent years, the improvement in heuristic methods has become under the name "local search methods" as well as there are Genetic Algorithm and Ant colony Algorithm. The following subsections describe various heuristic (approximate) methods for solving machine scheduling problems and discuss various parameters. That specifies the design of each heuristic, our discussion includes adaptations of known procedure as well as modification and extensions specifically suitable for the problem in this paper ($F_3/l_i,k_i/C_{max}$).

9.1. Neighborhood search methods

In neighborhood search methods we start from initial solution p and put it as a current solution, a current solution (p) is transformed into a new solution p′ according to some neighborhood structure, an acceptance rule decides whether the move from the current solution p to the transformed solution p′, if a move is accepted, then the transformed solution p′ replaces the previous solution p and becomes the current solution; otherwise the move rejected and the current solution is retained. This process is repeated until some termination criterion is satisfied. The acceptance rule is usually dependent on the objective function value of the current solution and its neighbor (13). The choice of suitable neighborhood is of significant importance, we introduce some of well-known neighborhoods for a permutation problem, where the set of feasible solution is given by the set of permutation of n-jobs :

- **Insert neighborhood;** in a permutation p= (p(1), p(2), …, p(n)), select an arbitrary job p(i) and shift it to position (J), when J > i or J < i this neighborhood some time referred to as a (shift neighborhood).
- **Swap neighborhood** ; in a permutation p= (p(1), p(2), …, p(n)), select two arbitrary jobs p(i) and p(j), (i ≠ j) and interchange them, this neighborhood sometime referred to as (pairwise interchange).

Before starting the description of these heuristic methods we need to define the following two terms (initial solution and stopping criterion).

*** Initial solution:**

The initial solution is the point from which the local search procedure is started, this could be a solution obtained from a heuristic or generated randomly, since a random solution may not satisfy the minimum of objective function. For our problem, the objective is the minimization of the makespan, so we can start from any of heuristics introduced in section (6). (H1, H2, H3, H4) since H4 is the best one as we were seen, thus the initial sequence is the sequence obtained by applying JR on the two artificial machine α and β

with processing time ($a_i+l_i+b_i+k_i$, $l_i+b_i+k_i+c_i$).

*** Stopping criterion: (14)**

The stopping criterion is the method used to terminate the search process; there are four common stopping criteria for local search algorithm:

1- A maximum number of iterations or solutions reached.
2- No improvement of the best solution for specified number of iterations.
3- Maximum C.P.U. time allowed for solving problem.
4- Fixed number of iterations.

The second criterion may be more efficient in speed, but since the number of iterations with no improvement will be effected by the complexity of solution space and problem size, so it is difficult to determine a number of iterations using (3) is partially captured in setting the maximum number of iterations i.e. (3 ≡ 1), therefore, the first stopping criterion is chosen by some researchers. In our work, we will use stopping criterion number (4), since we need to enhance the comparability of different local search heuristics and chose a fixed number of iterations that will make us unbiased for any of the local search heuristic methods.

9.1.1. Descent method (DM):

The Descent method is a simple form of neighborhood search methods in which only improving moves are allowed.

A number of issues have to be dealt with when (DM) procedure implemented:

1. **Initialization:**

The search has to be initialized with an initial solution. This solution can be constructed by some heuristic method or it can be chosen at random, the choice of this starting solution may greatly influence the quality of the final outcome, there is no guarantee that a good initial solution will lead to a near-optimal final solution. However the computational results of Osman and Potts (15) indicated that descent is slightly erratic unless a good starting solution is used. In our problem ($F_3/l_i,k_i/C_{max}$) we start from heuristic4 (H4) which present is in section (6).

2. **Generate of Neighborhood:**

There are many methods to generate a neighborhood; the two basic neighborhoods are (insert and swap) each of them is illustrated in (9.1). For the problem (F_3/l_i, k_i/C_{max}) we will use (swap) neighborhood.

3. **Stopping criterion:**

In this issue, as we see, there are four ways, but we will use a fixed number of iteration in more precisely we put that number equal to (**1000**) for solving our problem.

Although good choices for the above mentioned three issues can improve the performance of the descent method algorithm. The resulting solution is a local optimum, not necessarily a global optimum. A classical remedy for this draw-back is to perform multiple runs of the procedure starting from different initial solutions and to take the best

sequence as final solution such an approach is called (multi-start descent) [16]. In order to keep our unbiased for any method, so that we do not use the last note about multi-start, and stile use single initial solution.

9.1.2. Simulated Annealing (SA) method

Simulated annealing (SA) has its origin in statistical physics, where the process of cooling solids slowly until they reach a low energy state is called annealing. It was originally proposed by Metropolis et al. (17) and was first applied to combinatorial optimization problems by Kirkpatrick et al. (18). In such an algorithm, the sequence of the objective function values does not necessarily monotonically decrease. Starting with an initial sequence p, a neighbor p' is generated (usually randomly) in a certain neighborhood. Then the difference $\Delta = F(p')-F(p)$, in the values of the objective function F is calculated. When $\Delta <= 0$, sequence p' is accepted as the new starting solution for the next iteration. In the case of $\Delta>0$, sequence p' is accepted as new starting solution with probability $\exp(-\Delta/T)$, where T is a parameter known as the temperature. Typically, in the initial stages, the temperature is rather high so that escaping from a local optimum in the first iterations is rather easy. After having generated a certain number of sequences, the temperature usually decreases. Often, this is done by a geometric cooling scheme which we will also apply. In this case, the new temperature T^{new} is chosen such that $T^{new}=\lambda T^{old}$, where $0<\lambda<1$ and T^{old} denotes the old temperature. A possible stopping criterion would then be a cycle of a final temperature, which is sufficiently close to zero.

Determination (SA) algorithm parameters

To determine the parameters of (SA) for our problem ($F_3/l_i,k_i$ / C_{max}) we discuss the following issues:

a) Initialization
b) Neighborhood generation
c) Accepting move
d) Termination criterion

a) Initialization

- **initial solution**, we start from the sequence found by the heuristic (H4),which is described in section (6), to obtain the current solution (s) and compute the objective function value (makespan) of this sequence as a current value, although, theoretically there is no reason to start from a good initial solution. Due to the high probability to accept the deterioration in early stages, this good solution will be destroyed.
- **Initial temperature**, the second initial parameter we need to determine is the initial temperature value, basically the procedure start at (high) temperature where virtually all moves are accepted. Some researchers use a lower initial temperature because of using a good initial solution, another group of researchers used an abbreviated trial-annealing run in order to determine the initial temperature. In this paper we followed the last group of researchers by choosing better temperature value from a list of values after testing each one on the same set of examples, finally we chose (40^0) as an initial temperature value for (SA) .

b) neighborhood generation

The second issue is the neighborhood structure, different types of structures have been described, but we use here as in (DM) the (**swap**) neighborhood.

c) accepting move

Osman and Potts (15), show that the random neighborhood search method is better with the metropolis scheme $P(\Delta,T)=EXP(-\Delta/T)$. The probability of accepting an increase of objective function value depends on the size of this increase, the probability is small for a large increase. In our work, it is accepted that the moves decrease the objective function value, but for the moves that increase the objective function value by (Δ), the move will be accepted if $P(\Delta,T) = EXP(-\Delta/T) \geq R$. Where ($0<R<1$), otherwise the move will be rejected, where (T) decrease gradually by geometric cooling scheme $T=\lambda T$, ($0<\lambda<1$), in particular we put $\lambda= 0.999$.

d) Termination criterion:

It is sufficient to choose a fixed number of iteration, as we determine for (DM), this fixed number is equal to (1000).

9.1.3. Threshold Accepting (TA) method

Threshold accepting (TA) was originally proposed by Dueck and Scheuer (19) which can be regarded as a deterministic variant of simulated annealing. The idea is to accept moves with a non improving objective function value not with a certain probability, but only if the increase in the objective function value of the neighbor does not exceed a given threshold value V. The threshold value is usually rather large in the initial stages to allow the search an adequate covering of the solution space, but then it is reduced as the algorithm progresses, Glass and Potts (20). In our test, we considered a linear reduction of the value of V. We chose the initial threshold value V^0, after making a multiple run, (with a different threshold value V), on several examples, and then we chose the value (V^0), that gives maximum percent of decrease in the objective function of our problem. The solution in this method is based on the initial threshold value (V^0), and the number of iteration (14), we linearly reduced the threshold value such that the final cycle with constant V is performed for V=0. In addition to decreasing thresholds, there is an adjusted scheme which always starts with $V^0=0$. If during the last generated solutions no neighbor has been accepted, we increase the value $V=V+\Delta V$. If during the next generated solutions, again no neighbor has been accepted, we once more increase the value of V by ΔV. However, as soon as a neighbor has been accepted, we reset V=0. Such a refined scheme allows moves to solutions with worse objective function value only when it becomes difficult to find better neighbors and when the danger that the search stagnates in a local optimum increases.

Determination (TA) algorithm parameters

To determine the parameters of (TA) for our problem (F_3/l_i , k_i /C_{max}). Each of the following issues is discussed:

a) Initialization

Determine the initial solution as the above mentioned methods i.e. we chose the sequence which is found by heuristic (H4) which is described in section (6), and for the initial value of V we put (V=30), and the liner reduction (v = v – 0.01 v), and if it becomes close to zero we will reset it to (30).

b) Neighborhood generation

For the neighborhood we suggested the (swap) neighborhood as in previous methods.

c) Termination criterion

Stopping criterion for (TA) that chosen here is not differ from that use in the other local search methods i.e.(fixed number of iteration, which is equal to (1000).

9.1.4.Tabu Search (TS) method

Tabu search's (TS) origin dates back to the 1960s and 1970s and was proposed in its present form by Glover in (1989), (14). The majority of the applications of TS started in late 1980s (21). One of the main ideas of TS, as its name depicts, is its use of a flexible memory (tabu list) to tabu certain moves for a period of time. In every iteration of TS, a move will be instantly assigned to the tabu list when the move is chosen to lead the search from the current solution to its neighbor solution. This move will then not be chosen for a number of immediately succeeding iterations. This number of iterations is denoted as tabu list size, and the size is limited to a certain length. When the list has reached its specified length, the move that was assigned to the list earliest is released from the list and the most current move is inserted. With an appropriate design of the tabu list, TS is able to prevent cycling of the search and guide the search to the solution regions which have not been examined and approach to good solutions in the solution space. However, design of the tabu list may also prohibit the search to appealing solution regions. Next, we discuss some parameters affecting the performance of TS. The neighborhood size represents the number of candidate solutions to be evaluated in each iteration of the search process. Tabu search uses two common types of neighborhood size. The first kind is to evaluate all possible neighbors and select the best non-tabued solution in each iteration. This kind of examination may be suitable if the cardinality of the neighborhood is not too large. The quality of the solution obtained by this neighborhood examination may be good but the diversification capability of TS may be affected. The second type is to evaluate only a fixed number (only one is used here) of neighbors in one iteration.

Determination (TS) algorithm parameters

To determine the parameters of (TS) for our problem (F_3 / l_i , k_i / C_{max}) we discuss each of the following issues:

a) Initialization

Determine the initial solution as the above mentioned methods i.e. we chose the sequence, found by heuristic (H4).

b) Neighborhood generation,

For the neighborhood, we suggested the (swap) neighborhood as in previous methods.

c) Termination criterion

Stopping criterion for (TS) chosen here is the same as that use are in the previous local search methods i.e. (Fixed number of iteration, which equal to (1000).

9.2. Genetic Algorithm (GA)

Genetic Algorithms (GA) were originally proposed by John H. Holland (22). They are search algorithms that explore a solution space and mimic the biological evolution process.

Genetic algorithms work with the population of solution each solution is represented as a string the (GA) technique based on the mechanism of evolution. The solution space is usually represented by a population. New structures are generated by applying simple genetic operators such as (select, cross-over, and mutation). The members with higher fitness values (i.e., better objective function values) in the current population will have higher probability of being selected as parents, which is similar to Darwin's concept of survival of the fittest. The initial population is randomly generated, which means that the optimality of the final solution would not be guaranteed. Therefore, in the initial population, at least one solution having the minimum makespan (objective function of our problem) is included applying (select, cross-over and mutation), to generate new population and save the best solution in every generation. The best one from saved solutions becomes GA solution (14), the fitness value of a solution is a vector representing the function values (makespan). A parent is generated by selecting the best solutions from the current population. Then, solutions with good fitness values in each population are selected and recombined in each generation to produce a new offspring after applying the genetic operators for each new offspring we get a new population. We note that the mutation operation (for example) is based on the pairwise interchange (swap) of two jobs in the corresponding sequence. There are several applications of Genetic Algorithms (GA) have been widely applied to various fields since 1975.

9.2.1 Basic Stricture of Genetic Algorithm

The main components of a genetic algorithm are as follows (23) :

1) Solution Encoding

A chromosomal representation of solutions, (solution encoding). For the machine schedule problem, the natural permutation representation of a solution is a permutation of the integers 1,…,n, which defines the processing order of n jobs. Each chromosome is represented by such a scheduling solution, i.e., the natural permutation representation of a solution.

2) Initial Population

The creation of an initial population of chromosomes, (initial population).

In order to approximate an optimal solution as near as possible, the initial population of chromosomes is created by scheduling heuristic dispatching rules (heuristics methods), combined with random methods.

3) Fitness (evaluation)

The measurement of chromosome fitness is based on the objective function (fitness).When a population is generated, each chromosome is evaluated and its fitness is calculated for each chromosome. Finally each chromosome is assigned its fitness value along of the population size.

4) Selection

Natural selection of some chromosomes, by selection methods (according fitness value usually), chromosomes (parents) are selected from the population for combining to produce new chromosomes (children), i.e., for applying genetic operators.

5) Genetic Operators

Genetic Operators (crossover and mutation) applied to the chromosomes whose role is to create new members, i.e., children, in the population by crossing the genes of two chromosomes (crossover operators) or by modifying the genes of one chromosome (mutation operators):

a) **Crossover:**

The role of a crossover operator is to combine elements from two parent chromosomes to generate one or more child chromosomes.

b) **Mutation:**

The role of a mutation operator is to provide and maintain diversity in a population so that other operators can continue to work.

6) Replacement

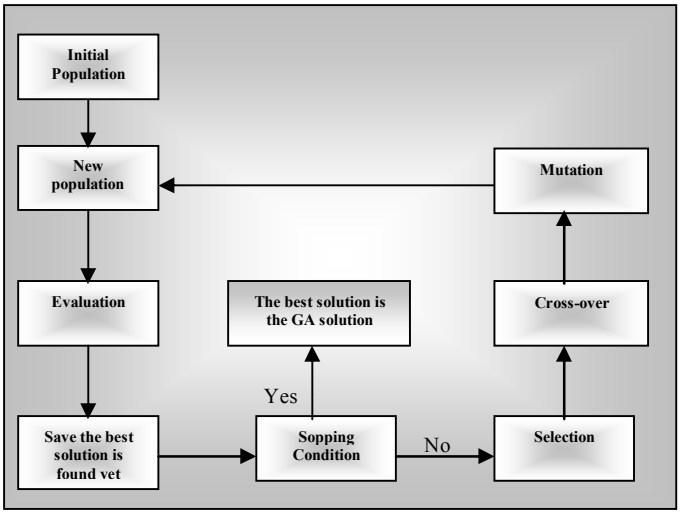
Natural selection of the members of the population, who will survive (replacement) is based on elitism. That is to keep the best chromosomes of the current population and their offspring. They will form a new population to survive into the next generation.

7) Parameter Selection

Natural convergence of the whole population that is globally improved at each step of the algorithm. For choosing suitable values of parameters such as population, size crossover and mutation.

The performance of a (GA) depends largely on the design of the above components and the choice of parameters such as population size, probabilities of genetic operators (i.e., crossover and mutation), and number of generations.

The following cycle give us the outline of (GA):



Genetic algorithm cycle

9.2.2. Genetic algorithm components

1)**Initial population**

The initial population can be generated at random or can be constructed by using problem-specific knowledge. Chen et al. (24) used specific construction heuristics for the flow shop problem to build their first population. They claim that a good initial population increases the efficiency of GA. Delia Croce et al. (25) select the solution for the initial population at random, but in order to speed up convergence, propose to choose an initial population partially produced with some quick heuristic. Inserting a high-fitness chromosome into the initial population is called (seeding) the success of the strategy is dependent on the availability of good starting solution; the large variation in the population size (m), used by different researches, ranging from a size of 20 Lee and Kim(26) to 300 Delia Croce et al. (25).

2) Selection

Selections to choose good candidate solutions from current population for the next generation i.e.(for generate the next population). The number of these candidate solutions (k) is controlled (determined) by the population size (m), which is selected in the initial steps of (GA).

3) Genetic Operators

a) Cross over

The crossover plays a role of exchanging information among chromosomes. It usually leads to an effective combination of partial solutions on other chromosomes, and accelerates the search procedure early in the generation. Partially matched crossover **(PMX)** is such an operator. Two crossover points are generated at random and the

segments in between define a matching section. This matching is used to affect a cross through position-by-position exchange operations. For example, with crossover points after the 3rd and 6th element:

	Parents:		Exchanging:		Restoring:	
P1	798|251|634	→	798|483|634	→	795|483|612	= Ch 1
P2	956|483|271		956|251|271		986|251|473	= Ch 2

In this example, the mapping is 2 ↔ 4, 5 ↔ 8 and 1 ↔ 3. Between the two crossover points, the sections are exchanged. To restore feasibility in the first child , the elements 8, 3 and 4 outside the section are replaced according to the matching. In the second child , the elements 5, 2 and 1 are replaced. A lot of papers refer to (**PMX**), for example Chen et al. (24).

Delia Croce et al. (25) use linear order crossover **(LOX)** This operator chooses two random crossover points. The elements in the cross section of parent 1 are removed from parent2 leaving some "holes" (shown with '.'). The holes are slid from the extremities towards the center until they reach the cross section. The cross section is then substituted with that of parent 1. The other child is obtained analogously.

	Parents	Holes	Sliding	Exchanging	
P1	798|251|634	79.|251|6..	792|…|516	792|483|516	Ch1
P2	956|483|271	9.6|483|.7.	964|…|837	964|251|837	Ch2

LOX tends mainly to respect relative positions between the elements and also, as far as possible, the absolute positions in the string. For example, the ordering of the first cross section (2,5,1) is completely destroyed in the first offspring by PMX. In the first offspring produced by LOX, the relative order, 2 before 5 and 1 and 5 before 1, is preserved.

There is another cross over scheme: (homogeneous mixture crossover) **HMX** (11), which is given by the mixture of the two parents uniformly by making a set (m) from genes, the odd position from the first parent and the even position from the second parent. Then separating genes without the repetition of the gene, since we read the set (m) from the left, if the gene j does not exist in the first child ch1 then keep it and put (0) in (m), otherwise we keep gene j in the second child ch2 and put (1)in (m), until (m) genes are exhausted. This way also gives two new children.

Parent	Mixture	Child
P1= 798251634	7 9 9 5 8 6 2 4 5 8 1 3 6 2 3 7 4 1	ch1 = 795862413
P2= 956483271	0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1	ch2 = 958623741

The rationale for this crossover is that it preserves the absolute positions taken from one parent, and the relative positions of those from the other parent. However, after a number of generations, the population has converged and crossover alone cannot improve the population anymore. A diversifying component is necessary which can be offered by mutation.

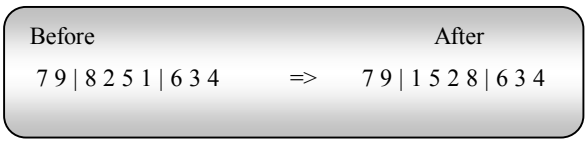
b) Mutation

Order-mutation: interchanges these two elements.

Position-mutation: places the second element before the first.

Order-mutation performs better than position-mutation. Other researches use the same techniques, but refer to them by more classical names, as that used in neighborhood search methods "swap" and "shifting".

Tate and Smith (27) use another form for mutation. They select two locations at random in a string and reverse the order of all elements within the substring bounded by the two selected elements.



4) Termination

Classically, the procedure stops when a fixed number of generations (or iterations) are executed. For example, Chen et al. (24) observe that the solutions become stable after twenty generations; therefore, they use 20 generations. Because of this fixed number of generations, it is possible that some generations at the end of the process are superfluous. To avoid this, the procedure can be terminated when the best solution in a population is not better than the previous population for a number of iterations. Lee and Kim (26) used this termination condition. There are other stopping criteria which terminate the procedure when the objective function values for the best and worst individuals in the population are equal (16). The algorithm of Lee and Kim (26) stops when the improvement of the average fitness value in one generation is less than 0.01% of the average fitness value in the preceding generation.

Determine the (GA) parameters

Initial population generation

We construct the initial population by using some individual solutions found by the four heuristics methods {H1, H2, H3, H4} which introduced in section (6) and the rest solutions are generated at a random. This technique was used by Reeves (28).

Selection

Selections to choose good solutions from current population. The number of this selected solutions (k) is controlled (determined) by the population size (m).

Genetic operator

- **Cross over**

Among the crossover rules that introduced here {PMX, LOX, HMX} we chose the last one (HMX) since it needs less computational time than the other rules which gives the same influence (play same role) of others rules.

- **Mutation**

We chose (swap mutation), or(Order-mutation) since it performs better than position-mutation, as we see after test them by multiple runs on several examples .

Population size and stopping condition

The efficiency of (GA) is dependent mainly on the (population size and stopping condition) parameters, since both of them are determine the speed of (GA) and the convergent to nearest optimal solution, so we should be determine them in more precisely. The most important question here is (how to determine these values?). We suggested that: the populating size (m) is chosen from the set A={40, 84, 180, 312}, as we mention in section (9.2.2) ranging from a size of (20 to 300), and the number of iteration is chosen from the set B={50, 100, 250, 500}. Then for each value of (A) solve same example along all values of (B), in this way we will have (4x4) values matrix and (4x4) times matrix for each example we solve it. From these examples and (other ones for different 'n') we get, the increase in the value of population size (m), often gives an enhance of the solutions, but the time associated with these values of 'm' (rather) is very high. In addition, we can note that increasing the number of iteration does not always give an improvement of the solution; moreover, this increase leads to an increase of the computational time. So we chose for the population (m = 180), and for number of iteration we have a hesitation between (100 and 250) but for unbiased between all heuristics methods we shall chose (100) as a number of iteration for (GA) to solve our problem.

9.3 Ant Colony Optimization (ACO)Algorithm

The Ant Colony Optimization (ACO) algorithm, originally introduced by Dorigo et al.(29), is a cooperative heuristic searching algorithm inspired by the ethological study on the behavior of ants. It was observed that ants – who lack sophisticated vision – could manage to establish the optimal path between their colony and the food source within a very short period of time. This is done by an indirect communication via the chemical substance, or pheromone, left by the ants on the paths. Though any single ant moves essentially at random, it will make a decision on its direction based by the “strength” of the pheromone trails that lie before it, where a higher amount of pheromone hints a better path. As an ant traverses a path, it reinforces that path with its own pheromone, which in turn creates an even larger amount of pheromone on those short trails, which makes those short trails more likely to be chosen by future ants (30).

9.3.1 Basic definition of (ACO)

The Main idea of the (ACO) is to keep a population or colony of (n) artificial ants that iteratively builds solution by continually applying a probabilistic decision policy (n) times until a solution is found. Ants that found a good solution mark their path through the decision space by putting some amount of pheromone on the edges of the path. Ants of the next iteration are attracted to the pheromone resulting in a higher probability to follow the already traversed good paths. In addition to the pheromone values, the ants will usually be guided by some problem specific heuristic for evaluating possible decisions regarding which direction to take along the way. In ACO algorithm, ants have a memory that stores visited components of their current path.

Apart from the construction of solutions and depositing of pheromone, the ACO incorporates other methods, pheromone evaporation, it causes the amount of pheromone on each edge to decrease over time. The important property of evaporation is that it prevents premature convergence to a sub-optimal solution. In this manner, the ACO has the capability of "forgetting" bad solutions, which favors the exploration of the search space (31).

9.3.2 Ant colony optimization(ACO) algorithm

(ACO) was suggested as a new heuristic method to solve optimization problems by Dorigo and Gambardella (32). The form of algorithm and functions is shown as follows. Each ant generates a complete solution by choosing the nodes according to a probabilistic state transition rule. The state transition rule is given in (1) is called a pseudorandom-proportional rule:

$$P^k(i,j) = \frac{(t_{i,j})^{\alpha} \quad (\eta_{i,j})^{\beta}}{\sum_{j \in N_i^k} (t_{i,j})^{\alpha} \quad (\eta_{i,j})^{\beta}} \quad \ldots(1)$$

Where t_{ij} is the amount of pheromone in edge ij, $\eta_{ij}=1/\delta_{ij}$ where δ_{ij} is the cost of edge ij, α and β are parameters that determine the relative importance of η versus t ,and N_i^k is the remaining node set of ant k based on moving from node i to build a feasible solution [33] .The parameters α, β are user defined parameters that determine the degree to which the pheromone is used versus the heuristic distance in deciding where to move. Setting $\beta = 0$ will result in only the pheromone information being used whereas if $\alpha=0$, only the heuristic information will be used (31). In either case in ACO, only the globally best ant that has built the best solution deposits pheromone in the graph. At the end of an iteration of the algorithm, once all the ants have built a solution, pheromone is added to the arcs used by the ant that found the best tour from the beginning of the trial. This updating rule is called the global updating rule of pheromone:

$$t_{ij} = (1-p)t_{ij} + p.\ \Delta t_{ij} \qquad \ldots. (2)$$

where $0 < p < 1$ is a pheromone decay parameter and Δt_{ij} equals to

$$\Delta t_{ij} = \begin{cases} \frac{1}{best\ cost} & if\ (i,j) \in best\ sequence \\ 0 & otherwise \end{cases} \quad \ldots (3)$$

In ACO, ants perform step-by-step pheromone updates using local updating rule of pheromone. These updates are performed to favor the emergence of other solutions than the best so far. The updates result in step-by-step reduction of the pheromone level of the visiting edges by each ant.

The local updating rule of pheromone is performed by applying the rule :

$$t_{ij} = (1 - \zeta).t_{ij} + \zeta.t_0 \quad \ldots. \quad (4)$$

t_0 is a small fixed value and $0 < \zeta < 1$ is the local evaporation of pheromone (33).
The ACO structure is shown in the following procedure:

1 Set pheromone trails to be small constant
2 **While** (termination condition not met)
3 Place each ant on initial node (its index usually)
4 **Repeat**
5 **For** each ant do
6 Chose next node by Apply State Transition Rule
7 **End For**
8 **Until** "each ant build one a solution"
9 Chose the best solution
10 Apply Local Update pheromone
11 Apply Global Update
12 **End While**

ACO algorithm procedure

9.3.3.New modification of ACO

In contracts of the local search algorithms, the ACO algorithm does not start from initial solution as we see, this will make ACO algorithm occasionally weaker than the other local search algorithms. In this paper we modified the ACO algorithm by making it start from a good heuristic method, doing this modification for ACO algorithm (the new procedure) is as follows:

Updating (global updating) the path (solution), found by the considered heuristic before starting the ACO algorithm procedure (old procedure). Particularly for our problem (F_3/l_i, k_i /C_{max}) problem, we use H4 heuristic as an initial solution. The following table (3) shows that the two solutions for each example found by both of the new and old procedure and also the number of best solution for each procedure, where

n= the number of jobs (problem size).
New pro. = the value of ACO that use the new procedure.
Old pro. = the value of ACO that use the old procedure.
No. best = number of best solution of examples with associated procedure.

Table -3:Comparsion between new and old ACO algorithm

n / ex	50		100		200		300	
	New pro.	**Old pro.**	**New pro.**	**Old pro.**	**New pro.**	**Old pro.**	**New pro.**	**Old pro.**
1	2908	3225	5270	5669	11153	11142	15266	15691
2	3034	3188	5777	6213	10486	11184	15769	16080
3	2819	3315	5945	6559	10739	11554	15215	16268
4	4265	4460	7101	7606	11809	13133	17096	18581
5	4969	5316	7285	8066	11724	14167	16717	19025
6	6189	6716	8370	8954	12359	14110	17494	19874
7	6747	7295	9181	9984	13643	14965	17794	20758
8	8114	8098	10167	11333	14085	16227	19220	21450
9	8917	9603	11027	12269	15814	17348	19139	22557
10	9670	9730	11872	13271	16883	18438	20600	22782
No. best	**9**	**1**	**10**	**0**	**9**	**1**	**10**	**0**

It is clear that the algorithm that uses new procedure (start from a good initial solution) is better than the algorithm that uses old procedure (with out initial solution). We will use the ACO algorithm that uses the new procedure i.e. the ACO algorithm that starts from a good initial solution.

9.3.4. Determination ACO parameters

At first, according to Dorigo and Gambardella (32) the initial values for parameters are set to the following values:

(i) global evaporation coefficient of pheromone, p=0.1;

(ii) local evaporation coefficient of pheromone, ζ=0.1;

(iii) Pheromone initial amount on edges, t_{ij}=0.000005 for all i and j;

(iv)The number of ants in the colony of the problem is considered as twice as the number of jobs i.e. (2n).

(v) The fixed initial value of t_0 is t_0 = 0.012.

Furthermore, by considering this fact that in the proposed algorithm, the length (cost) of arcs does not have a meaning therefore by supposing β=0, the length effect of edges is omitted in ACO.

In our problem, we consider the number of ants in the colony which is equal to (4n) when (n) is the number of jobs. Then the best parameters values are experimentally adjusted. For this purpose, the parameters values based on the best parameters values previously found, the parameter values are iterated incrementally and then the algorithm runs ten times. After this step the best value was chosen and then the problem is solved with these best parameters.

- P parameter. The value of P is chosen from the set (A) where (A) is the set of all iterations from 0 to 1 by increments of 0.05, after ten runs on each element of (A) we determining the value of (P), (that is supposed) as P=0.25.
- ζ parameter. Based on P = 0.25, the value of ζ also chosen from the set (A), for determining ζ value, based on ten runs on each element of (A). The most favorable value is supposed as $\zeta = 0.2$
- t_0 parameter Based on P = 0.25, and ζ =0.2, the value of t_0 is based on set of iterations from 0 to 0.0004 by steps of 0.00002. The best value equal t_0= 0.

9.4 Computational results and comparison

9.4.1 Test problems for the local search algorithms

To discuss the comparative effectiveness of introduced local search algorithms, we generate two classes of test problems (examples) :

Class 1 : is the same as the one generated in section (8.1)

Class 2 : this class of examples had been constructed in special design such that these examples are satisfied the conditions given in section (5), as a solvable special cases, for this class of examples the optimal solution is known, and we use this class to compare the solutions of local search algorithms with the optimal solution, on problem of size "n" where n $\in$ {25, 50, 100, 200, 300}, and for each value of "n" we generate (10) examples.

9.4.2 Computational results

All local search algorithms in this paper (Decent Method, Simulation Annealing, Threshold Accepting, Tabu Search, Genetic Algorithm, Ant Colony algorithm), are coded in Matlab language, and run on Pentium (IV). In our computational, we use the condition that: if the solution of an example with "n" jobs for any algorithm is not appear after (600) seconds, from its run; then this example is unsolved and this algorithm is active until the problem of size "n". This criteria used by Stoppler and Bierwrith (34), their time out was (900) seconds.

9.4.3 Comparative effective of local search algorithms

Table (4), shows for each algorithm, how many it can catch the optimal value for each value of "n" (problem size). Where n $\in$ {5, 7, 9, 12, 25, 50, 100, 200, 300}. The optimal solution for examples of small size n $\in$ {5, 7, 9, 12}, was found by using BAB algorithm, and for problems of large size n $\in$ {25, 50, 100, 200, 300}, we use class 2 of examples (i.e. the optimal solution was known for these examples).

Table (4), show for each algorithm, how many it can catch the optimal value for each value of "n" (problem size). Where n $\in$ {5, 7, 9, 12, 25, 50, 100, 200, 300}.

Table (5) shows the average time of (10) examples for each algorithm and each (n), which is relative to table (4).

Table-4: show for each algorithm, how many it can catch the optimal value for each value of "n" (problem size) Where n ∈ {5, 7, 9, 12, 25, 50, 100, 200, 300}.

n	GA	ACO	TS	TA	SA	DM
5	10	8	10	10	10	10
7	10	4	9	10	10	10
9	10	4	10	7	10	8
10	10	1	9	8	8	8
12	9	2	8	6	7	10
25	10	2	5	4	4	6
50	9	0	3	0	5	5
100	9	0	0	0	0	0
200	9	0	2	0	0	1
300	7	0	0	0	0	0
sum	93/ 100	21/ 100	56/ 100	45/ 100	54/ 100	58/ 100

Table -5: shows the average time of (10) examples for each algorithm and each (n)

n	GA	ACO	TS	TA	SA	DM
5	2.1928	0.0088	0.1981	0.1855	0.1856	0.1853
7	2.7979	0.0073	0.2374	0.2063	0.2075	0.2057
9	3.5908	0.0116	0.3381	0.2497	0.2387	0.2254
10	3.7237	0.0145	0.292	0.2326	0.2307	0.2293
12	4.3437	0.0199	0.3319	0.2491	0.26	0.2486
25	9.0498	0.0893	0.7457	0.353	0.356	0.3511
50	20.9784	0.3931	1.3979	0.5869	0.5931	0.5809
100	55.7584	1.9353	2.7111	1.0338	1.0297	1.0285
200	159.7052	11.1326	5.5498	2.0098	2.0066	1.9969
300	322.4879	34.9933	9.1273	3.1579	3.1656	3.1601

Table-6:shows for each algorithm,how many it can catch the best value for each value of "n" (problem size)

n	GA	ACO	TS	TA	SA	DM
50	7	0	2	1	1	2
100	9	0	0	0	0	2
200	10	0	0	0	0	0
400	10	2	2	2	3	3
700	*	3	7	4	4	4
1000	*	*	7	7	6	8
1500	*	*	3	5	5	6
2000	*	*	8	7	8	8

2500	*	*	6	7	4	5
3000	*	*	8	8	9	8
sum	36/ 40	5/ 50	45/ 100	41/ 100	40/ 100	45/ 100

Table (6), shows for each algorithm, how many it can catch the best value for each value of "n" (problem size). Where n ∈ {50, 100, 200, 400, 700, 1000, 1500, 2000, 3000}.
* = refer to the unsolved examples.
Table (7) shows the average time of (10) examples for each algorithm and each (n), which is relative to table (6).
*= refer to the unsolved examples

Table -7: shows the average time of (10) examples for each algorithm and each (n), which is relative to table (6).

n	GA	ACO	TS	TA	SA	DM
50	20.6209	0.4054	0.7089	0.5507	0.5672	0.5658
100	53.8523	2.0715	1.3265	0.9987	0.9957	0.9987
200	158.8443	11.038	3.0125	1.9776	1.9745	1.9776
400	525.5447	80.9357	7.1421	4.2421	4.2233	4.2292
700	*	419.5833	17.2946	9.2349	9.299	9.2601
1000	*	*	30.3904	15.8181	15.8405	15.8059
1500	*	*	53.4493	27.6583	27.7519	27.4455
2000	*	*	83.7002	45.2615	43.6872	43.7598
2500	*	*	108.8085	59.7966	59.6727	59.7674
3000	*	*	141.2573	80.4619	80.7	80.0922

9.4.4. Efficiency of local search algorithms

The computational times of all algorithms for the ($F_3/l_i,k_i/C_{max}$) problem, with our modifications on these algorithms, are approximately the same (except the Genetic algorithm and Ant colony algorithm), since the computational time of (GA) is very large as compared with the computational time of neighborhood search methods. Indeed this difference of time comes from the way that uses to generate the new sequence in each method, where the neighborhood search algorithms use swap neighborhood, which needs small time to perform its procedure. While, the Genetic algorithm use genetic operators (cross-over and mutation), which consume large time to perform its procedure; but the computational time of Ant colony algorithm is gradually increasing with the increase of problem size, since the procedure of (ACO), dependent on the accumulated pheromone on each node. According to the condition that had been given in section (9.4.2), we have the following table (8), which gives the activity of local search algorithms, (i.e. give the maximum number of jobs "n" that the local search algorithms can solve the ($F_3/l_i,k_i/C_{max}$) problem with reasonable time:

Table -8:Activity of the Local Search Algorithms

Algorithm	active until (maximum no. of jobs)
GA	n = 400
ACO	n = 700
TS	n = 4500
TA	n = 7000
SA	n = 7000
DM	n = 7000

9.4.5 Summary of Experimental evaluation of local search algorithms

Now we summarize the finding empirical evaluation of introduced local search methods,

- Choice of good neighborhood significantly influences the quality of results, for the problem under consideration, in this paper we use both of swap and insert neighborhood schemes. From a comparative study of the difference type of neighborhood search methods, (Descent Method, Simulation Annealing, Threshold Accepting and Tabu Search), we found that for our problem ($F_3/l_i,k_i/C_{max}$), use of swap neighborhood scheme generally produced the best results, than the use of insert neighborhood scheme. Also for Genetic Algorithm we use swap neighborhood scheme and insert neighborhood scheme as a mutation operator; the use of swap neighborhood scheme leads to better results than the insert neighborhood scheme.
- For Descent method, multi-start descent method performed better than single start descent method.
- For Simulation annealing, the (SA) algorithm that start from high temperature value is worked best than the (SA) algorithm that start from low or random temperature value.
- For Threshold Accepting, the use of linear reduced scheme make the (TA) algorithm more efficient than the use of linear increase scheme, which start from zero as an initial threshold value, and then growth linearly.
- For Tabu Search, the (TS) algorithm that evaluate all possible neighborhood and select the best one in each iteration is worse than the (TA) algorithm that evaluate only one neighborhood in each iteration.
- In Genetic algorithm, Homogeneous cross-over (HMX), make the procedure of (GA) runs rather quickly, thereby requiring computational time less than the other cross-over schemes the Partially matched crossover (PMX), the linear order crossover (LOX).
- For Ant Colony Optimization, start from initial solution; will make the ACO worked best. And the use only the pheromone information is decrease the value of solution in ACO algorithm, by certain percentage; less than the solution value of ACO algorithm that

use both of pheromone information and heuristic objective function value, or only heuristic objective function value.

- From comparative study of the introduced local search algorithms and precise vision of above all tables, we found that for the ($F_3/l_i,k_i/C_{max}$) problem, the Genetic algorithm is obtained the best results. For the problem of size less than or equal (400) jobs, in this case, the recommended (GA) is that use (180) as a population size, and (100) as a cyclic iteration, with swap neighborhood scheme as a mutation operator. For the problem of size large than the (400) jobs, we found that the Descent method and Tabu search algorithm produced best results. Due to considerably the difference of computational time between the above two methods, we suggested that the (DM) is recommended. Indeed this difference of computational time comes since the (TS), lose too much time in creating and updating the Tabu list. In other hand the Simulation Annealing and Threshold Accepting algorithms, have the objective function values close to that found by (DM) and (TS) algorithms.

10.Conclusion

This paper, discusses the **Exact** and **Local search methods**. For the problem of scheduling 'n' jobs on three machines (A, B, and C) flow shop with transportation times between the machines, where the machine can process one job at a time. And each job 'i' is transported by l_i and k_i from A to B and from B to C respectively, the ($F_3/l_i,k_i/C_{max}$) problem is known as NP-hard problem. Two results concerning optimality of two solvable special cases are presented. Also, we found a good four heuristic methods and used them as upper bounds (UB) for (BAB) algorithm. We introduced two lower bounds (LB-I- and LB-II-). We conclude that: the lower bound LB-II- is the best one for (F_3/l_i, k_i/C_{max}) problem. This result is found by testing these lower bounds on large set of problems, which is generated randomly.

The paper, developed and tested various local search heuristic methods, designed experiments and analyzed the effects of various parameters used in the local search methods. Our experimental results indicated that: some of the local search heuristic algorithms can solve ($F_3/l_i,k_i/C_{max}$) problem of size $\leq$ (7000) jobs in reasonable time. Also we found that the Genetic algorithm is the best algorithm for the ($F_3/l_i,k_i/C_{max}$) problems of size less than or equal to (400) jobs. And for problem of large size the Descent Method was recommended.

11. Future work

Some suggestions for future research are described as follows:

First, the extension of the propose of the exact methods for ($F_3/l_i,k_i/C_{max}$) problem by driving a good lower bound or using the dominances rule in branch and bound algorithms.

Second, using the local search heuristic should be explored for finding an improvement potential of various polynomially bounded scheduling heuristic.

REFERENCES

1. Johnson S.M., "Optimal Two and Three Stage Production Schedule with Setup Time Included", Nav.Res.Log.Quart.1(1) (1954).
2. Conway R.W., Maxwell W.L., and Miller L.W. "Theory of Scheduling" Addison Wesley, Reading, MA. (1967).
3. S. REZAHEJAZI And S.SAGHAFIAN, " Flow shop-scheduling problems with makespan criterion:a review",International Journal of Production Research, 43, (14) 15July :2895–2929(2005).
4. Bonney, M.C.and Gundry, S.W., Solution to the constrained flowshop sequencing problem. Operat.Res.Quart., 24:869–883(1976).
5. King,J.R. and Spachis,A.S., Heuristics for flow-shop scheduling. Int. J.Prod.Res., 18, :345–357(1980).
6. Rajendran,C., "A heuristic for scheduling in flow shop and flow line-based manufacturing cell withmulti-criteria" . Int.J. Prod.Res., 3: 2541–2558(1994).
7. Hurink J., Knust S. "Flow shop problems with transportation time and single robot", Osanbruck University, Osanbruck Reihp, April (1998).
8. Belman R., Eesogbue A.O., Nabeshima I., "Mathematical Aspects of Scheduling and Application", pergamon press, Oxford, New York, Toronto, Sydney, Paris, (1982).
9. Baker, K.R "Introduction to Sequencing and Scheduling " Wiley New York, (1974).
10. Franch, S. "Sequencing and Scheduling an Introduction to Mathematics of Job Shop" , John Wiley & Sons, New York (1982).
11. Mohammed, H. A.A. "Using Genetic and local search algorithms as a tool for providing optimality for job scheduling", M.Sc. Thesis, College of Science, University of Al-Mustansiriyah, (2005).
12. Al-Maraashi, N. A.A. "Three machine flow shop problem with transportation time ". M.Sc. Thesis. College of Science, University of Al-Mustansiriyah (2006).
13. Graham R.L., Lawler E.L., Lenstra J.K., and Rinnooy Kan A.H.G. "Optimization and Approximation in Deterministic Sequence and Schedule: a Survey Annals of Discrete Mathematics, 5, (1979).
14. Gupta J.N.D., Hennig K., Werner F., "Local search heuristics for two-stage flow shop problems with secondary criterion ", Ball stat university, Muncie, IN43,306, USA. Pergamon computer and operation research 29, :123-149,(2002).
15. Osman I. H., and Potts C. N., "Simulated annealing for permutation flow shop scheduling ", OMEGA ,17:551-557 (1989).
16. Crauwels, H. "A comparative study of local search methods for one machine sequence problem". Ph.D. thesis Katholieke University, Heverlee. Belgium (1998).
17. Metropolis M., Rosenbluth A., Rosenbluth M., Teller A., Teller M., "Equation of state calculations by fast computing machine", Journal of chemical physics 21: 1087-92 (1953).
18. Kirkpatrick S., Gelatt Jr. CD., Vecchi MP., "Optimization by simulated annealing ", Science 220:671-80 (1983).

19. Dueck G, Scheuer T. "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing". Journal of Computational Physics; 90:161-75 (1990).
20. Glass C.A., Potts C.N. "A comparison of local search methods for flow shop scheduling". Annals of Operations Research; 63:489-509 (1996).
21. Reeves CR. "Modern Heuristic Techniques for Combinatorial Problems". John Wiley and Sons, Inc, New York, (1993).
22. Holland J. H. "Adaptation in Natural and Artificial Systems". Ann Arbor, University of Michigan Press, (1975).
23. Liu N., Mohamed A. Abdelrahman, and Srini Ramaswamy," A Genetic Algorithm for the Single Machine Total Weighted Tardiness Problem", Tennessee Technological University, Cookeville, TN 38505, USA, (2003).
24. Chen C.L., Vempati V.S., and Aljaber N. , " An application of genetic algorithms for flow shop problems" European Journal of operation research,80:389-396 (1995).
25. Della Croce F., Tadi R. and Valta G., "A Genetic algorithm for job shop problems ". computer and operation research, 22 :15-40 (1995).
26. Lee J.K. and Kim Y.D., "Search heuristic for resource constrained project scheduling", Journal of the operation research society 47, :678-689 (1996).
27. Tate D.M., and Simth A.E., "A genetic approach to the quadratic assignment problem". computer and operation. research. 22:73-83 (1995).
28. Reeves C.R.," A Genetic algorithm for flow shop sequencing"computer and operation research, 22: 5-13 (1995).
29. Dorigo M., Maniezzo V., and Colorni A., "A positive feedback as a search strategy ", Milan, Italy. tech. Rep.: 91-016. (1991).
30. Gang Wang, Wenrui Gong , Ryan Kastner, " Instruction Scheduling Using (MAX - MIN) Ant System Optimization", Department of Electrical and Computer Engin - eering, University of California at Santa Barbara,Chicago,Illinois,USA.:17–19, April (2005).
31. Ventresca M., and Ombuki B.M.. "Ant Colony Optimization for Job Shop Scheduling Problem" , Brock university Canada, L25, 3A1, (2004).
32. Dorigo M., and Gambardlla, L.M., "Ant algorithms for discrete optimization", Massachusetts Institute of technology, artificial life 5: 137-172 (1999).
33. Keivan G. and Fahimeh M., "ACS - TS: train scheduling using Ant Colony system", Journal of applied mathematics and design sciences. Article ID (95060) :1-28, (2006).
34. Stöppler S. and Brierwirth C. " The Application of parallel Genetic algorithm to the n/m/p/C_{max} flow shop problem" , University of Bremen <C13,f@dhbrrz 41.bet.net>