MOEA/D with Tabu Search for Multiobjective Permutation Flow Shop Scheduling Problems

Ahmad Alhindi, *Student Member, IEEE,* and Qingfu Zhang, *Senior Member, IEEE,*

***Abstract*— Multiobjective Evolutionary Algorithm based on Decomposition (MOEA/D) decomposes a multiobjective optimisation problem into a number of single-objective problems and optimises them in a collaborative manner. This paper investigates how to use Tabu Search (TS), a well-studied single objective heuristic to enhance MOEA/D performance. In our proposed approach, the TS is applied to these subproblems with the aim to escape from local optimal solutions. The experimental studies have shown that MOEA/D with TS outperforms the classical MOEA/D on multiobjective permutation flow shop scheduling problems. It also have demonstrated that use of problem specific knowledge can significantly improve the algorithm performance.**

***Index Terms*— Decomposition, multiobjective optimisation, Tabu search.**

I. Introduction

A Multiobjective optimisation problem (MOP) can be stated as follow:

$$\begin{aligned} \text{minimise} \quad & F(x) = (f_1(x), \ldots, f_m(x)) \\ \text{subject to} \quad & x \in \mathcal{D} \end{aligned} \tag{1}$$

where x is a potential solution, $\mathcal{D}$ is the discrete search space, and *F(x)* consists of m scalar objective functions $f_1(x), \ldots, f_m(x)$. Very often, an improvement in one objective will cause a degradation of another. No single solution can optimise all of the objectives at the same time. A decision maker has to balance these objectives in an optimal way. The concept of Pareto optimality is commonly used to best trade off solutions.

Given two solutions $x, y \in \mathcal{D}$, x is said to dominate y if and only if $f_i(x) \leq f_i(y)$ for every i and $f_j(x) < f_j(y)$ for at least one index $j \in \{1, \ldots, m\}$. x^* is called Pareto optimal to (1) if no other solution dominates x. The set of all the Pareto optimal solutions is called the *Pareto set* (PS), the set $\{F(x)|x \in PS\}$ is called the *Pareto front* (PF).

The goal of multiobjective evolutionary algorithms (MOEAs) is to produce a number of solutions to approximate the PF in a single run. Such approximation can be useful for a decision maker to understand a problem and make a final decision. Along with Pareto dominance-based MOEAs and hypervolume-based MOEAs, decomposition-based MOEAs (MOEA/D [1]) have been widely accepted as a major approach in the area of multiobjective evolutionary computation. MOEA/D decomposes the MOP into a number of subproblems. Each subproblem can be a single-objective problem or a simpler multiobjective problem; then a population based method is used to solve these subproblems in a collaborative way.

Some advanced single-objective methods of local search, such as iterative local search (ILS), greedy randomized adaptive search (GRASP) [2], tabu search (TS) [3] and guided local search (GLS) [4] are able to escape local optimal solutions and obtain a reasonably good solution. These single-objective local search methods have been well studied and documented. It is worthwhile to study how to use them in MOEAs. Since MOEA/D optimises a number of single-objective optimisation subproblems, it provides a very natural framework for using single-objective local search techniques. Actually, some efforts have been made to hybridise single-objective search methods with MOEA/D. Examples include MOEA/D with ant colony optimisation (ACO) [5], MOEA/D with simulated annealing (SA) [6] and MOEA/D with guided local search (GLS) [7].

This paper proposes a combination of MOEA/D with tabu search (TS), called MOEA/D-TS. Our major motivation is to use TS to help MOEA/D escape from local Pareto optimal solutions. TS is an advanced neighbourhood search that uses attributive memory structures to guide the search away from local optima, i.e., a short-term memory to prevent the reversal of recent moves and longer-term frequency memory to reinforce attractive components. This type of memory records information about solution attributes that change in moving from one solution to another.

In MOEA/D-TS, subproblems are optimised in parallel. Each subproblem has a different weight vector (i.e., search direction), a memory and a single solution. For a subproblem trapped in a local optimal solution, TS starts with its current solution x and uses its memory information as the guided information to explore the neighbourhood $\mathcal{N}(x)$ of solution x. Then, TS selects the best neighbouring solution $x' \in \mathcal{N}(x)$ to replace the current solution x even if it does not improve the objective function value with the aim of escaping from the attraction region of the current solution. This process is repeated for a number of iterations or until the local optimisation of TS does not yield a better solution.

The rest of this paper is organised as follows. Section II presents the proposed MOEA/D-TS algorithm. Section III describes multiobjective permutation flow shop scheduling

Ahmad Alhindi is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, U.K. (email: aalhin@essex.ac.uk). He is currently on scholarship from the College of Computer & Information Systems, Umm Al-Qura University, Makkah, Saudi Arabia (e-mail:ahhindi@uqu.edu.sa)

Qingfu Zhang is with the Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong (e-mail: qingfu.zhang@cityu.edu.hk). He is currently on leave from the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, U.K. (e-mail: qzhang@essex.ac.uk).

problem. In Section IV, the experimental results are presented and discussed. Section V demonstrated the effect of some algorithmic components on the performance of MOEA/D-TS. Finally, Section VI concludes the paper and outlines some avenues for future research.

II. The Proposed Algorithm: MOEA/D-TS

This section introduces the major ideas and techniques used in our proposed algorithm.

A. Algorithmic Framework

MOEA/D-TS first decomposes the MOP into N single objective subproblems by choosing N weight vectors $\lambda^1, \ldots, \lambda^N$. In this paper, we adopt the weighted sum approach [8] to construct subproblems. The objective function of subproblem is:

$$\begin{aligned} \text{minimise} \quad & g^{ws}(x|\lambda) = \sum_{i=1}^{m} \lambda_i f_i(x) \\ \text{subject to} \quad & x \in \mathcal{D} \end{aligned} \tag{2}$$

where $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a weight vector with $\sum_{i=1}^{m} \lambda_i = 1$ and $\lambda_i \geq 0$ for all $i = 1, \ldots, m$. Moreover, a subproblem has tabu memory structure η, which stores its learned knowledge by the TS during the local search phase.

When the quality of a solution is very poor, the application of local search on it may waste computational effort [9]. Thus, local search should be applied to only good solutions. Motivated by this observation, MOEA/D-TS introduces a dynamic selection scheme of initial solutions (i.e., subproblems) for local search. For choosing promising subproblems, we define and compute a utility π^i for each subproblem i [10]. TS selects subproblems based on their utilities.

During the search, MOEA/D-TS maintains:

- a population of N points $x^1, \ldots, x^N \in \mathcal{D}$, where x^i is the current solution to the ith subproblem;
- $FV^1, \ldots, FV^N$, where FV^i is the F-value of x^i, i.e. $FV^i = F(x^i)$ for each $i = 1, \ldots, N$;
- $\pi^1, \ldots, \pi^N$: where π^i is the utility of subproblem i.
- $\eta^1, \ldots, \eta^N$: where η^i is the tabu memory for subproblem i, storing its learned knowledge.
- EP, it is an external archive containing all the nondominated solutions found so far.
- gen: the current generation number.

MOEA/D-TS works as follows:

Step 1: Initialization

Step 1.1 Compute the Euclidean distances between any two weight vectors and then find the T closest weight vectors to each weight vector. For each $i = 1, \ldots, N$, set $B(i) = i_1, \ldots, i_T$ where $\lambda^{i_1}, \ldots, \lambda^{i_t}$ are the T closest weight vectors to λ^i.

Step 1.2 Generate an initial population $x^1, \ldots, x^N$ by uniformly randomly sampling from the search space.

Step 1.3 Set EP= ϕ, $gen = 0$ and $\pi^i = 1$ for all $i = 1, \ldots, N$.

Step 2: Genetic Search: for $i = 1, \ldots, N$ do

Step 2.1 Reproduction: Randomly select two indexes k, l from $B(i)$, and then generate a new solution y from x^k and x^l by using genetic operators.

Step 2.2 Update of Neighbouring Solutions: For each index $j \in B(i)$, if $g(y|\lambda^j) \leq g(x^j|\lambda^j)$, then set $x^j = y$ and $FV^j = F(y)$.

Step 2.3 Update of EP: If no vectors in EP dominate $F(y^i)$, add $F(y^i)$ to EP and remove from EP all the vectors dominated by $F(y^i)$.

Step 3: $gen = gen + 1$.

Step 4: Termination: If a problem specific stopping condition is met, stop and output EP.

Step 5: Local Search: If gen is a multiplication of 25, do the following:

Step 5.1 Update Utility: Compute Δ^i, the relative decrease of the objcetive for each subproblem i during the last 25 generations, update

$$\pi_i = \begin{cases} 1, & \text{if } \Delta^i > 0.001 \\ (0.95 + 0.05\frac{\Delta}{0.001})\pi^i, & \text{otherwise;} \end{cases} \tag{3}$$

Step 5.2 Selection of Subproblems: By using 10-tournament selection based on π^i, select $[\frac{N}{5}] - m$ indexes and add them to I.

Step 5.3 Tabu Search: For each $i \in I$, conduct a single objective Tabu search starting from x^i for subproblem i to generate a new solution y^i. Then update $B(i)$ and EP with y^i.

Step 6: Return to **Step 2**

Since major computation overhead is caused by TS, therefore, we only conduct TS every 25 generations and on some selected subproblem in Step 5. In **Step 5.1**, the relative decrease Δ is defined as

$$\Delta = \frac{\text{old function value-new function value}}{\text{old function value}} \tag{4}$$

and computed for each subproblem i in order to update its utility. In 10-tournament selection in **Step 5.2**, the index with the highest π^i value from 10 uniformly randomly selected indexes are chosen to enter I. We do this selection $[\frac{N}{5}] - m$ times.

TS (Step 5.3) detailed in Procedure 1 starts from the current solution x^i of the i-th subproblem and uses its memory information η^i as the guided information during the search. Then, TS explores the neighbourhood $\mathcal{N}(x^i)$ of x^i. Any solution $x' \in \mathcal{N}(x^i)$ can be reached from x^i by a single *move* from x'. TS moves from a solution to its best admissible neighbour, even if this move deteriorates the objective function. To avoid cycling, TS uses the tabu memory η^i to forbid or tabu (for a number of iterations) any moves which can take the search back to some solutions recently explored. The Tabu status of a move is overridden when certain criteria (aspiration criteria) are satisfied. An example of aspiration criteria is to allow moves that produce solutions better than the currently-known best solutions.

In the above top-level description, the details of some steps are problem-specific. In the following, we take the multiobjetive permutation flow shop scheduling problem as an example to show how these steps can be implemented. We would like to point out that our implementation is not unique. There are several possible ways to instantiate the above framework.

Procedure 1 TabuSearch(x^i, w^i, η^i)

Inputs:
x^i:*Current solution.*
w^i:*Search direction.*
η^i:*Tabu memory.*
Output: *The best solution found* S^{best}
1: $S \leftarrow x^i$; *//initialise starting solution*
2: $S^{best} \leftarrow x^i$; *//initialise best solution*
3: $AL \leftarrow g(x^i|w^i)$; *//initialise Aspiration level*
4: **repeat**
5: Generate neighbour solutions $N^* \subset \mathcal{N}(S)$
6: Find best $S^* \in N^*$
7: **if** move S to S^* is not on η^i or $g(S^*|w^i) < AL$ **then**
8: Accept move and update the best solution.
9: Update tabu memory and aspiration level.
10: **end if**
11: **if** $F(S^*)$ is not dominated by$F(S)$ **then**
12: Update external population EP
13: **end if**
14: **until** stopping condition is satisfied
15: **return** best solution

III. The Permutation Flow Shop Scheduling Problems

A. Problem Formulation

This paper considers the permutation flow shop scheduling problem (PFSP). The PFSP schedules n jobs with given processing times on m machines where the sequence of processing a job on all machines is identical and unidirectional for each job. All jobs are available at time zero. Each job can only be processed on at most one machine and each machine can process only one job at any time. Pre-emption is not allowed, i.e., once the processing of a job has started on a machine, it must be completed without interruption at that machine. Different jobs have the same processing order on all machines.

Definition 1. (PFSP): *Given a set of* n *jobs,* $\{J_1, \ldots, J_n\}$, m *machines,* $\{M_1, \ldots, M_m\}$, *and a* $m \times n$ *matrix* $P = [p_{ij}]$ *such that* $p_{i,j}$ *denote the processing time of job* j *on machine* i. *The* PFSP *is to find an optimal permutation* π *of* n *jobs processed on* m *machines, subject to feasibility constraints.*

B. Optimisation Objectives

The multiobjective PFSP in this paper is to minimise the total completion time, the total tardiness, and the total flow time. These three objectives are formally defined below.

1) Makespan: This objective is the completion time of the last job (i.e., the makespan, denoted by C_{max}). In order to find the makespan of a job schedule π, one needs to compute the completion time for the n jobs in all the k machines. Given a permutation $\pi = (\pi(1), \ldots, \pi(n))$,the completion time $C_{i,\pi(j)}$ of job $\pi(j)$ on a machine M_i can be computed using the following set of recursive equations.

$$\begin{aligned} &C_{1,\pi(j)} = \textstyle\sum_{j=1}^{m} p_{1,\pi(j)} \qquad j = 1, \ldots, n \\ &C_{i,\pi(1)} = \textstyle\sum_{i=1}^{m} p_{i,\pi(1)} \qquad i = 1, \ldots, m \\ &C_{i,\pi(j)} = \max\{C_{i,\pi(j-1)}, C_{i-1,\pi(j)}\} + p_{i,\pi(j)} \\ &\quad i = 2, \ldots, m; \\ &\quad j = 2, \ldots, n \end{aligned} \tag{5}$$

Then, the makespan is given by

$$C_{max}(\pi) = \max\{C_{1,\pi(1)}, \ldots, C_{i,\pi(j)}\}$$

Finally, the first objective is defined as

$$\min C_{max}(\pi) \tag{6}$$

2) Maximum Tardiness: For the PFSP with due date, the tardiness is the deadline for the completion of a job. The tardiness T_j of job j is defined as:

$$T_j = \max\{C_j - d_j, 0\}$$

where C_j and d_j is the completion time and due date of job j, respectively.

The second objective is then defined by

$$\min T_{max}(\pi) \tag{7}$$

with

$$T_{max}(\pi) = \max\{T_{\pi(1)}, \ldots, T_{\pi(n)}\}$$

3) Total Flow Time: The total flow time F is equal to the sum of completion times of jobs. This is an important performance measure in flow shop scheduling. The third objective is defined as

$$\min F(\pi) \tag{8}$$

with

$$F(\pi) = \sum_{j=1}^{n} C_{\pi(j)}$$

C. Structural Properties

It is well known that algorithm performances can be improved by exploiting problem specific properties. Several structural properties have been studied for the PFSP. In [11], Nowicki and Smutnicki have introduced block properties and successfully applied them to a tabu search algorithm.

The description of block properties requires the notion of critical path and block. Thus, we introduce the definition of critical path and block in section III-C.1 first and then the block properties.

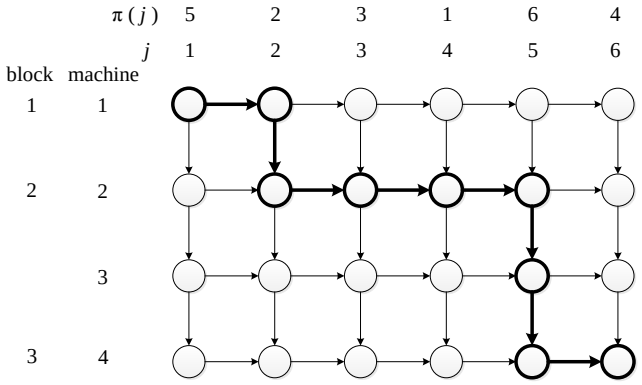


Fig. 1: The network $N(\pi)$ for $m = 4$, $n = 6$ and $\pi = (5, 2, 3, 1, 6, 4)$. The critical path is marked by thin lines;$u^* = (2, 5, 5)$

1) Critical Path and Block: Consider a network $N(\pi)$ with vertex valuations for a permutation $\pi \in \mathcal{D}$. Each vertex $(i, j) \in N(\pi)$ represents the job $\pi(j)$ on machine i and its valuations is the processing time $p_{i,\pi(j)}$. Each path $u \in N(\pi)$ from $(1, 1)$ to (m, n) is represented by a sequence $(u_0, u_1, \ldots, u_{m-1}, u_m)$ with $u_0 = 1$ and $u_m = n$ satisfying $u_0 \leq u_1 \ldots \leq u_{m-1} \leq u_m$, and is made of vertices $(1, u_0), \ldots, (1, u_1), (2, u_1), \ldots, (2, u_2), \ldots, (m, u_{m-1}), \ldots, (m, u_m)$. The length of the path $l(u)$, is given by the sum of the valuations of all vertices of the path. Mathematically, it is expressed as

$$l(u) = \sum_{i=1}^{m} \sum_{j=u_{i-1}}^{u_j} p_{i,\pi(j)} \tag{9}$$

Definition 2 (Critical Path). *A path $u^* = (u_0^*, u_1^*, \ldots, u_m^*)$ is called a* critical path *of π if it is the longest path in $N(\pi)$, i.e. $l(u^*) = \arg\max_{u \in N(\pi)}(l(u))$*

Definition 3 (Block). *Based on the critical path u^*, a sequence of jobs $B_k = (\pi(u_{k-1}^*), \pi(u_{k-1}^* + 1), \ldots, \pi(u_k^*)$ is called the kth* block *in π, $k = 1, 2, \ldots, m$. Next, the kth* internal block *is defined as a subsequence of B_k:*

$$B_k^* = \begin{cases} B_k - \{\pi(u_1^*)\}, & if \;\; k = 1; \\ B_k - \{\pi(u_{k-1}^*), \pi(u_k^*)\}, & if \;\; 1 < k < m; \\ B_k - \{\pi(u_{m-1}^*)\}, & if \;\; k = m. \end{cases} \tag{10}$$

It worth noting that by the definition the last job in block B_k is simultaneously the first in its neighbour $B_{k+1}, k = 1, 2, \ldots, m-1$.

As an example, Fig.1 shows a schedule of $n = 6$ jobs and $m = 4$ machines. The permutation $\pi = (5, 2, 3, 1, 6, 4)$ and its critical path $u^* = (2, 5, 5)$ which generate three blocks: $B_1 = (5, 2)$, $B_2 = (2, 3, 1, 6)$ and $B_3 = (6, 4)$, and three relevant internal blocks $B_1^* = (5)$, $B_2^* = (3, 1)$ and $B_3^* = (4)$.

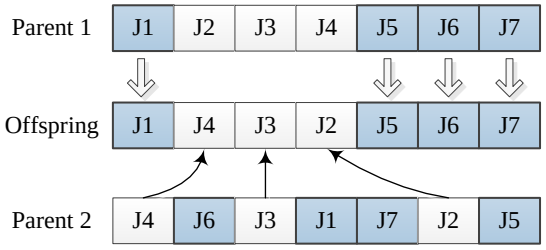


Fig. 2: Two point crossover operator

2) Block Properties: Block properties (BPs) are some general properties associated with the graph representation of scheduling problems with the makespane criterion [12]. This section introduce two BPs proposed by Nowicki [11] and Grabowski [13] [14] that are associated with the neighbourhood structure.

Block Property 1 [11] *Shifting a job within the internal block does not generate a better neighbour.*

Block Property 2 [13] [15] *Suppose π_v is generated by move $v = (x, y)$, where jobs $\pi(x)$ and $\pi(y)$ are in the p-th and l-th internal blocks of $\pi(x)$, respectively. Then it has*

$$C_{max}(\pi_v) \geq C_{max}(\pi) + p_{\pi(x)l} - p_{\pi(x)p} \tag{11}$$

IV. Experimental Studies

A. The Implementation of MOEA/D-TS for PFSP

In order to apply the MOEA/D-TS to the PFSP, the following components need to be defined.

1) Solution representations: There are several forms of representing candidate solutions for scheduling problems.This paper represents a candidate solutions as a permutation or sequence of n job $\pi = (\pi(1), \ldots, \pi(n))$, which makes it easy to maintain the feasibility of solutions throughout genetic search and local search.

2) Genetic operators: Since the PFSP is a sequencing problem with n job, various genetic operators proposed for travelling salesman problems and other scheduling problems are applicable. In [16], Murata and Ishibuchi have conducted a study of various genetic operators for single-objective PFSP to minimise the makespan. In their study, the two-point crossover and insertion mutation operators were regarded as the best genetic operators for this problem. For this reason, we adopt these genetic operators, which are illustrated in Fig.2 and Fig.3, respectively.

3) Tabu Search: In order to use TS in the proposed framework of MOEA/D-TS, moves, neighbourhood, tabu memory (its structure and management), aspiration criterion, and diversification must be defined.

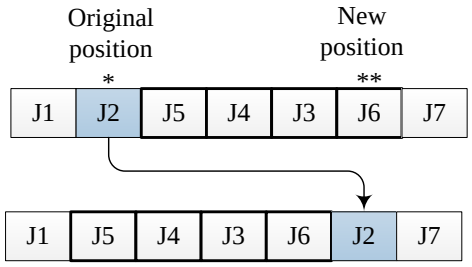


Fig. 3: Insert operation, which is used by genetic search as mutation operator and by local search as neighbourhood structure.

Moves and neighbourhood: The insert move operates on a permutation π of jobs. It removes a job placed at a position in this permutation and inserts it in another position in the permutation. More precisely, let $v = (a, b)$ be a pair of position in the permutation π, $a, b \in 1, 2, \ldots, n, a \neq b$. The pair $v = (a, b)$ defines a move in π. Consequently, the move v generates a neighbouring permutation π_v from π in the following way

$$\pi = (\pi(1), \ldots, \pi(a-1), \pi(a), \pi(a+1), \ldots, \pi(b-1), \pi(b), \pi(b+1)), \ldots, \pi(n)) \quad (12)$$

$$\pi_v = \begin{cases} (\pi(1), \ldots, \pi(a-1), \pi(a+1), \ldots, \\ \pi(b-1), \pi(b), \pi(a), \pi(b+1)), \ldots, \pi(n)), & \text{if } a < b \\ (\pi(1), \ldots, \pi(a-1), \pi(b), \pi(a), \pi(a+1), \\ \ldots, \pi(b-1), \pi(b+1)), \ldots, \pi(n)). & \text{if } a > b \end{cases} \quad (13)$$

The insertion neighbourhood of π consists of all the neighbouring permutations π_v obtained by a move from a given set U, and denoted as $\mathcal{N}(U, \pi) = \pi_v \in U$. Typically, the "original complete" insertion neighbourhood is generated by the move set $U = \{(a, b) | a, b \in \{1, 2, \ldots, n\}, b \notin \{a, a-1\}\}$ of size $(n-1)^2$. It is worth noting that the condition $b \notin \{a, a-1\}$ is added to avoid redundancy of moves. Such a large-size set would assist TS to avoid being trapped in a bad local optimum, however, its sizes drastically increases with the number of jobs. It is quite time consuming to evaluate all moves especially when we repeatedly evaluate neighbourhoods in TS.

Structural properties described in section III-C can assist us to reduce the original complete neighbourhood as follow:

- According to property 1, generating neighbours by shifting a job within an internal block are not interesting (note we are only interested in blocks of size greater than or equal to two). Consequently, the original complete insertion move set U can be reduced by removing such non-improving moves. Hence, the computation time is further decreased.
- According to property 2, when generating neighbours by shifting job between blocks, we can define and obtain a lower bound LB on the makespan C_{max} for neighbour solutions π_v, which defined as $LB(\pi_v) = C_{max}(\pi) + p_{\pi(a)l} - p_{\pi(a)p}$. Here obviously, if $p_{\pi(a)l} \geq p_{\pi(a)p}$, then $LB(\pi_v) \geq C_{max}(\pi)$, which implies that $C_{max}(\pi_v) \geq C_{max}(\pi)$. As a result, we can know that π_v is not better than π without explicitly evaluating π_v. Therefore, non-promising neighbours can be excluded and the computational effort for the search of neighbourhood is reduced.

Obviously, utilising these structural properties will improve the algorithm performance. The computation time will decrease and the search toward promising regions of the search space will be promoted.

Tabu Memory and its Structure: Tabu memory, denoted by η, is an essential element in TS. TS makes a systematic use of it to exploit knowledge beyond that contained in the objective function and the neighbourhood $\mathcal{N}$. Short-term and long-term memory are employed in the form of recency based and frequency based memory in this work. Short-term memory contains information that to some extend forbids the search from returning to a previously visited solution during the recent past seach, while long-term memory contains information that helps achieve global diversification of the search.

Because each move consists of (a, b) pair of positions, we represent the tabu memory by using a square matrix of order n illustrated in Fig.4; where n is the number of jobs. The upper triangle of the matrix is allocated for recency-memory to store tabu tenure value of a $v = (a, b)$, while the lower triangle matrix is allocated for frequency-memory to record the number of times the $v = (a, b)$ is encountered.

Since TS conducts on various subproblems with different local search directions, it might be good to use a different tabu memory η^i for each subproblem i.

Managment of Tabu Memory: Each time a move v is performed to go from solution π to π_v by removing a job at position a and inserting it at position b, the pair (x, y) of jobs is added to memory. The values of the pair (x, y) of jobs are determined as follow:

$$(x, y) = \begin{cases} (\pi(a), \pi(a+1)), & \text{if } a > b \\ (\pi(a-1), \pi(a)), & \text{if } b < a \end{cases} \quad (14)$$

This tabu mechanism is based on the idea that after removing a job from a position to another, any move that would restore the relative order of the job and its former neighbour should be declared tabu for number of iterations.

Each time the pair (x, y) is added to the memory, the corresponding element in the upper triangle of η is set to the current iteration number plus tabu tenure k, while the corresponding element in the lower triangle is increment by 1. At any moment, it is easy to verify if a given move is tabu or not by simply comparing the current iteration number with that recorded in the the upper triangle of matrix η.

The Aspiration criterion: The above tabu mechanism is sufficient to prevent the algorithm from being trapped in

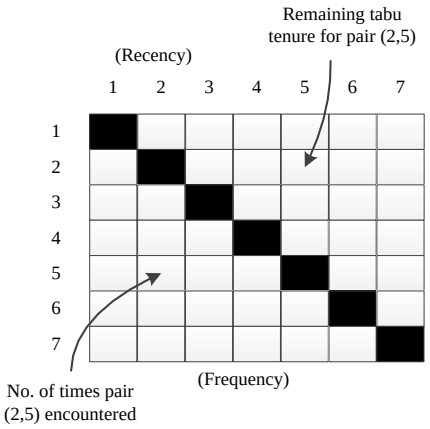


Fig. 4: Tabu memory structure. Upper triangle is used to store recency information while lower triangle is used to store frequency information.

short-term cycling. Meanwhile, such a mechanism may forbid solutions that are not yet visited. To fix this shortcoming, a standard and simple aspiration criterion is introduced. A tabu move is allowed if the move leads to a solution whose evaluation is better than that of the best solution found so far by the algorithm.

Diversification: Like other local search algorithms, TS tends to spend most of its time in a restricted region of the search space. Therefore, it is required to diversify the search process, i.e., force the search toward unexplored regions of the search space. The use of frequency information is used to conduct diversification, which integrates diversification directly into the regular searching process. To achieve this we replace a move value with the following penalty function:

$$MoveValue' = MoveValue + d \times Penalty \qquad (15)$$

where $Penalty$ value is frequency information obtained from the lower triangle of memory η, and d is adjustable diversification parameter. Larger d value correspond to more diversification and vice versa.

We refer interesting readers to [17], [18] for extensive discussion on various diversification strategies for TS.

B. Parameter Settings

The initial population is generated at random and the population size N is regulated by a parameter H. More precisely, $\lambda^1, \ldots, \lambda^N$ are all the weight vectors in which each individual weight takes a value from $\left\{\frac{0}{H}, \frac{1}{H}, \cdots, \frac{H}{H}\right\}$. Therefore, the number of such vectors is $N = C_{H+m-1}^{m-1}$. Table I lists the value of N and H. The setting of T, which determines the number of weight vectors in the neighbourhood of each weight vector, follows the recommendation in [19]. T is set to 3 and 10 for 2-objective and 3-objective test problems, respectively. All Pareto optimal solutions obtained during the genetic and local searches are recorded in an external population EP.

In MOEA/D-TS, local search is used every 25 generations and based on the proposed dynamic selection scheme in II, each fitness comparison performed inside the local search procedure is considered as an evaluation for fair comparison. For tabu search, the number of iteration inside TS is set to 30, the value of the penalty factor d is set to be 7 and the tabu tenure k is randomly and uniformly selected within a range $t^{min} = 3$ and $t^{max} = 20$. For genetic search, the crossover probability P_c and the mutation probability P_m are 0.85 and 0.5, respectively.

The maximum number of evaluation $maxEval$, listed in Table I, is used as stopping condition for each algorithms. It is worthwhile noting that large test problems need more computational workload because the size of the search space exponentially increases with the number of jobs. Thus, we perform computation experiments with more computation workload for 80-job test problems.

TABLE I: Parameter Setting for the Test Problems of the PFSP

Test Problem	$maxEval$	$N(H)$	T
2/20	1.0×10^5	200 (199)	3
2/40	1.0×10^5	200 (199)	3
2/60	1.0×10^5	200 (199)	3
2/80	2.0×10^5	200 (199)	3
3/20	1.0×10^5	300 (23)	10
3/40	1.0×10^5	300 (23)	10
3/60	1.0×10^5	300 (23)	10
3/80	2.0×10^5	300 (23)	10

C. Assessment Metrics

The following performance metrics are used in assessing the performance of the algorithms in our experimental studies.

1) Set Coverage (C-metric): Let A and B be two approximations to the PF of a MOP; the $\mathcal{C}(A, B)$ metric is defined as the proportion of solutions in B that are dominated by at least the sole solution in A:

$$\mathcal{C}(A, B) = \frac{|\{b \in B | \exists a \in A : a \,\text{dominates}\, b\}|}{|B|} \qquad (16)$$

The value of $\mathcal{C}(A, B)$=1 means that all solutions in B are dominated by some solutions in A, and $\mathcal{C}(A, B)$=0 implies that no solution in B is dominated by a solution in A.

2) Inverted Generational Distance (IGD-metric): Let P^* be a set of uniformly distributed points in the objective space along the PF. Let A be an approximation to the PF. The IGD from P^* to A is defined as follows:

$$IGD(P^*, A) = \frac{\sum_{v \in p^*} d(v, A)}{|P^*|} \qquad (17)$$

where $d(v, A)$ is the minimum Euclidean distance between v and the points in A. The smaller the value of IGD, the closer the solutions are to the Pareto optimal front. The IGD requires the true Pareto optimal fronts in advance. Thus, we used the composite Pareto optimal front of each test problem

in [9] [1] as a true Pareto optimal front in calculation of IGD in our experiments.

D. Results and Discussion

MOEA/D-TS is compared with a classical MOEA/D on 2-objective and 3-objective PFSP test instances. A set of nine test instances, with 20, 40, 60 and 80 jobs on 20 machine, proposed in [9][1] are used in our experimental studies. All the experiments have been carried out on identical computers (Intel Core i7 2.80GHz CPU and 8GB RAM). The programming language is Java. All the statistics are based on 50 independent runs.

TABLE II: Best, average and worst values of the IGD-metric of the solutions found by MOEA/D and MOEA/D-TS

Test	MOEA/D			MOEA/D-TS		
Problem	Best	Avg.	Worst	Best	Avg	Worst
2/20	25.9	45.6	86	1.6	7.1	12.2
2/40	97.8	134.5	168.7	25.8	42.8	59.3
2/60	185.7	260.4	407.3	53.9	81.5	127.2
2/80	135.4	213.2	290.4	23.77	48.22	75.97
3/20	63.7	106.8	172.1	17.6	26.7	35.8
3/40	314.4	490.8	1108.2	109.8	162.9	270.9
3/60	620.5	1175.8	2827.8	254.6	390.6	590.8
3/80	1014.8	2071.7	3259.9	454.5	749.9	1218.4

TABLE III: Average set coverage between MOEA/D *(A)* and MOEA/D-TS *(B)*

Test Problem	$\mathcal{C}(A,B)$	$\mathcal{C}(B,A)$
2/20	0.008	0.789
2/40	0.004	0.982
2/60	0.0	1.0
2/80	0.0	1.0
3/20	0.025	0.767
3/40	0.007	0.953
3/60	0.011	0.930
3/80	0.049	0.946

TABLE IV: The average number of the solutions found by MOEA/D and MOEA/D-TS. The numbers in parentheses represent the standard deviation.

Test Problem	MOEA/D	MOEA/D-TS
2/20	19.4(3.3)	39.1(2.6)
2/40	22(5.9)	52.1(7.5)
2/60	19.4(4.5)	48.8(6.5)
2/80	18.5(5.8)	39.3(5.2)
3/20	122.6(23.6)	402.9(31.2)
3/40	131(33.5)	580.5(57.1)
3/60	114.4(27.8)	559.3(61.2)
3/80	128.9(30.8)	579.6(87.9)

Table II summarises the best, average and worst values of IGD-metric of the final approximations obtained by each algorithm over 50 runs for each test instance. Table III shows the means of the $\mathcal{C}$-metric values of the final approximations obtained by the two algorithms. Table IV summarizes the number of non-dominated solutions obtained by the two algorithms. Fig. 5 plots the distribution of the final approximation with the lowest IGD value among 50 runs of each algorithm for each bi-objectives test instance.

We make the following remarks:

- Table II shows that the final non-dominated solutions obtained by MOEA/D-TS is better than those obtained by MOEA/D in terms of the IGD-metric for all the eight test instances. The results reflect the extend of enhancement MOEA/D-TS made to that MOEA/D and the ability of TS to escape Pareto local optimal area and obtain solutions closer to the PF with good spread over the PF. Taking instance 2/60 as example, this instance has an IGD value of 260.4 when the MOEA/D was used and 81.5 when the MOEA/D-TS was used.
- It is evident from Table III that the final solutions obtained by MOEA/D-TS are better than those obtained by MOEA/D, in term of $\mathcal{C}$-metric, for all the test instances. For instance, on average 93% of the final solutions obtained by MOEA/D on the 3/60 prolem instance are covered by those generated by MOEA/D-TS, while only 1.1% vice versa.
- We can see from Table IV that MOEA/D-TS is inferior to the classical MOEA/D in term of the number of obtained solutions for all the test problems. These results suggest that MOEA/D-TS tends to find more solutions with higher quality than MOEA/D.
- Fig.5 clearly indicates the difference between the final approximations obtained by the two algorithms on bi-objective test problems with 20, 40, 60 and 80 jobs.5. The results shown in this figure are consistent with the observation on the IGD and Set Coverages performance metrics. Moreover, Fig.5 reveals that the larger the number of decision variable is, the larger the difference between MOEA/D and MOEA/D-TS, which implies the search ability of MOEA/D-TS in large search spaces.

Overall, we can claim that MOEA/D-TS is more efficient and can produce better approximations than MOEA/D on these PFSP test instances.

V. Effect of Algorithmic Components and Sensitivity Analysis in MOEA/D-TS

A. Effect of Algorithmic Components

1) Effect of longer term structure: Diversification rules that drive the search into new regions is automatically created in TS (to some extent) by short-term memory functions, but the diversification is particularly reinforced by longer term memory. The short-term memory alone may be enough to achieve solution superior to those found by conventional local search methods, but long-term structures are often necessary for solving harder problems. This is why we incorporate long-term memory into MOEA/D-TS to help subproblems in avoiding being trap in local Pareto optimal solutions.

[1]The set of reference solutions & test problems are obtained from: http://www.ie.osakafu-u.ac.jp/~hisaoi/ci_lab_e/research/pdf_file/multiobjective/MOGLS

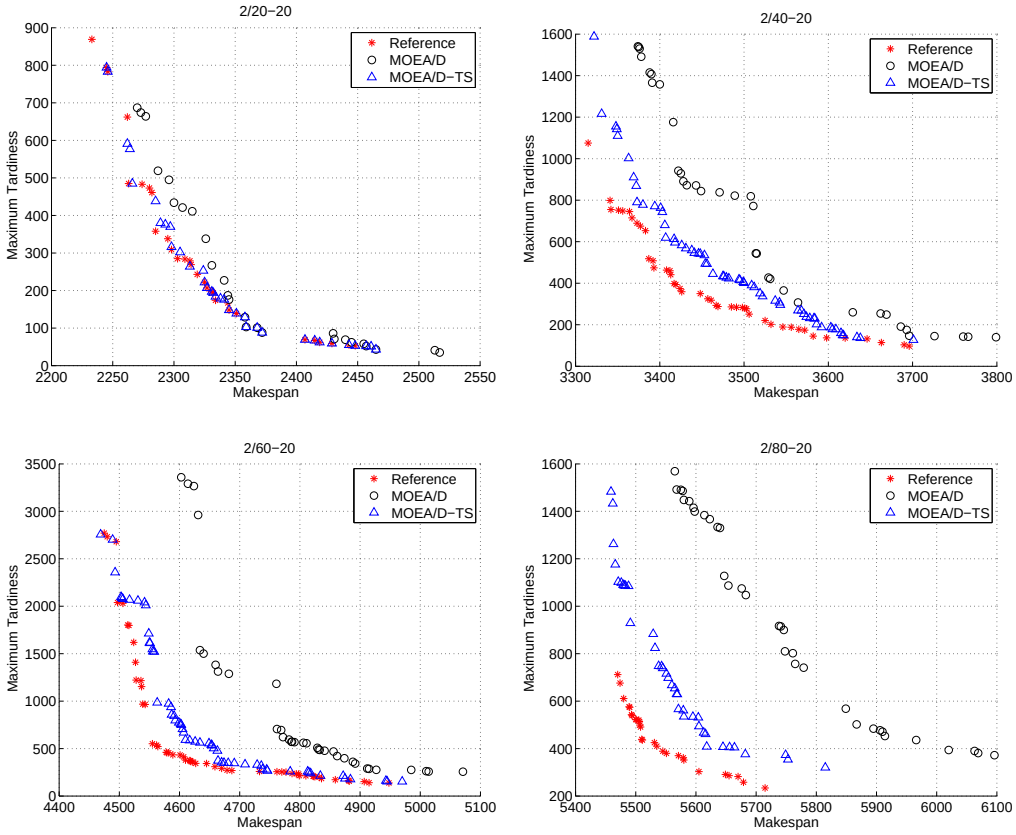


Fig. 5: Plot of non-dominated solutions with the lowest IGD value found by the two algorithms in 50 runs on bi-objective test instances with 20, 40, 60 and 80 jobs.

TABLE V: Comparison of three measures of the solutions found by MOEA/D-TS with and without long-term memory (LTM) on 80-job problem. The number in parentheses represent the standard deviation.

Measure	MOEA/D-TS	
	with LTM	without LTM
IGD-metric	48.22(8.2)	86.66(14.3)
$\mathcal{C}$-metric	0.413	0.389
$\|EP\|$	31.95(6.2)	26.1(6.7)

TABLE VI: Comparison of three measures of the solutions found by MOEA/D-TS with and without block properties (BPs) on 40-job problem. The number in parentheses represent the standard deviation.

Measure	MOEA/D-TS	
	without PBs	with PBs
IGD-metric	102.5(12.4)	51.7(10.5)
$\mathcal{C}$-metric	0.025	0.926
$\|EP\|$	31.55(7.3)	50.8(5.9)

To study the effect of using longer term structure, in terms of the quality and diversity of solutions, we have tried, on 80-job test instance, two variants of MOEA/D-TS with the same parameter settings except the incorporation of long-term memory. We have found that, on average among 50 independent runs as shown in Table V, MOEA/D-TS with long-term memory obtained lower IGD-metric values, therefore the final solutions are very good approximation to the PF. Moreover, the average number of solutions obtained is increased when using long-term memory. This indicate that the search has driven toward unexplored region in the search space.

2) Effect of structural properties: Does problem-specific knowledge such as structural properties has a significant impact on the performance of MOEA/D-TS? To answer this question, we have tried, on 2-objective 40-job test instance, two variants of MOEA/D-TS with the same parameter settings in Section IV-B except the use of structural properties. We have found that on average among 50 independent runs as shown in Table VI MOEA/D-TS using structural properties has lowered the IGD value, therefore the final obtained solutions are a very good approximation to the PF. This is consistent with the observation on $\mathcal{C}$-metric, where on average 92% of the final solutions generated by MOEA/D-TS using original insertion are dominated by those generated by MOEA/D-TS using structural properties, and only 2% vice versa. Moreover, MOEA/D-TS with block properties is inferior to MOEA/D-TS without in term of the number of obtained solutions for the bi-objective 40-job instance in Table VI. Clearly, these advantages come from incorpo-

rating structural properties. This means that incorporating such problem knowledge is very necessary in MOEA/D-TS. Therefore, we can conclude that problem knowledge help to improve the performance of the proposed MOEA/D-TS algorithm.

3) Choice of Scalarizing Function: The choice of an appropriate scalarizing function plays a very important role in MOEA/D. Several studies (e.g. [20], [21]) have demonstrated the effect of the choice of scalarizing functions on MOEA/D.

In this section, we examine the dependency of the performance of TS on the specification of a scalar function. We use the empirical attainment function (EAF) [22] to visually identify the algorithms' behaviour graphically. We compare two versions of MOEA/D-TS, one using the weighted sum approach of (2) and the other one using the weighted min-max approach [8]. For the latter, the scalar objective function of subproblem is:

$$\begin{aligned} \text{minimise} \quad & g^{ws}(x|\lambda) = \max_{i\in\{1,\ldots,m\}} \lambda_i f_i(x) \\ \text{subject to} \quad & x \in \mathcal{D} \end{aligned} \tag{18}$$

where λ is the same as of (2). We use the weighted min-max approach because it has a similar property to that of weighted Tchebycheff approach, which need a reference point [6].

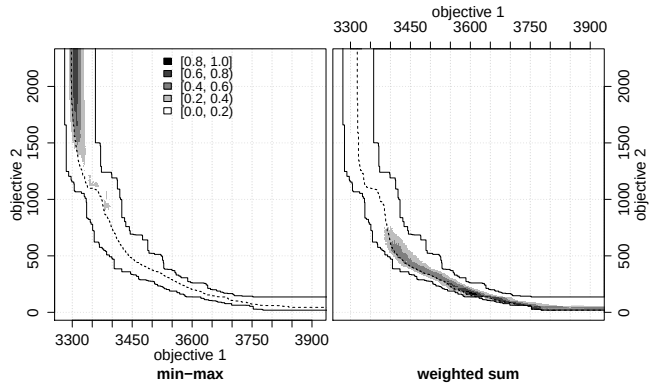


Fig. 6: Location of differences between min-max (left) and weighted sum (right) scalarizing functions for the bi-objective 40-job test problem.

Fig. 6 shows the location of differences between the EAFs over 100 runs of MOEA/D-TS with each scalarizing function. In each plot, the lower line corresponds to the best attainment surface associated to the solutions found with the minimum propbability, any solution below it was never attained. The upper line corresponds to the worst attainment surface associated to the solutions dominated with probability one, the region above this line was always attained. Thus, any difference would be within these two lines. The dashed line corresponds to the median attainment surface, which is given to facilitate the comparison of the two sides of the plot.

We can observe from Fig. 6 that MOEA/D-TS using min-max (left) performs better towards the minimisation of the second objective, whereas MOEA/D-TS using weighted sum performs better in the centre and towards the minimisation of first objective. Clearly, the search strategies behave differently with the change of the scalarizing function.

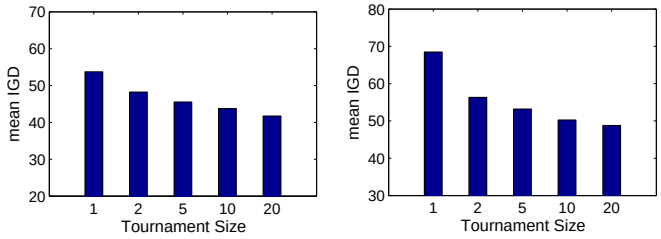
In general, the results presented indicate a strong dependency between the performance of TS and the choice of scalarizing functions. Therefore, these dependencies need to be taken into account when designing and implementing local search methods for MOEA/D.

4) Use of Utility for Selecting Local Search Solutions: In MOEA/D, there is no selection strategy to help identify good solutions (i.e. subproblems) for local search. A straight forward strategy is the random selection of subproblems for the application of local search. The random selection may lead to the selection of inappropriate initial solutions. In this paper, we have proposed the use of utility π to help select good solutions for the application of local search.

In this section, we demonstrate the improvement of MOEA/D-TS due to the proposed utility strategy. We compared two versions of MOEA/D-TS on instances with 40 and 80 jobs. The mean IGD values obtained using the random based selection are 50.97 and 65.09. While the utility based selection obtained 41.76 and 49.42. Clearly, MOEA/D-TS with the utility based selection perform better than the MOEA/D-TS without.

B. Sensitivity Analysis

1) The Size of Tournament Selection: In this paper, we use the tournament selection based on the utility function for choosing initial solutions for local search. In this section, we study the influence of the increase in the selection pressure (i.e., the increase in the tournament size) on the performance of MOEA/D-TS. We tested MOEA/D-TS with tournament sizes of $1, 2, 10$ and 20 on test instances with 40 and 80 jobs. Experimental results are summarised in Fig. 7.



(a) 40-job instance. (b) 80-job instance.

Fig. 7: Effect of the specification of the tournament size.

From this figure, we can see that the performance of MOEA/D-TS was improved by increasing the selection pressure of initial solutions for local search. This is attributed to the selection of appropriate initial solution for local search at different search stages. Note that, when the tournament size was specified as 1, initial solutions were randomly chosen from the population. In this case, the performance of MOEA/D-TS was degraded and this reflects the importance of utility function in selection of local search solutions.

2) Population Size (N) and the Number of Uniform Weight Vectors (H): In MOEA/D-TS, the population size N is

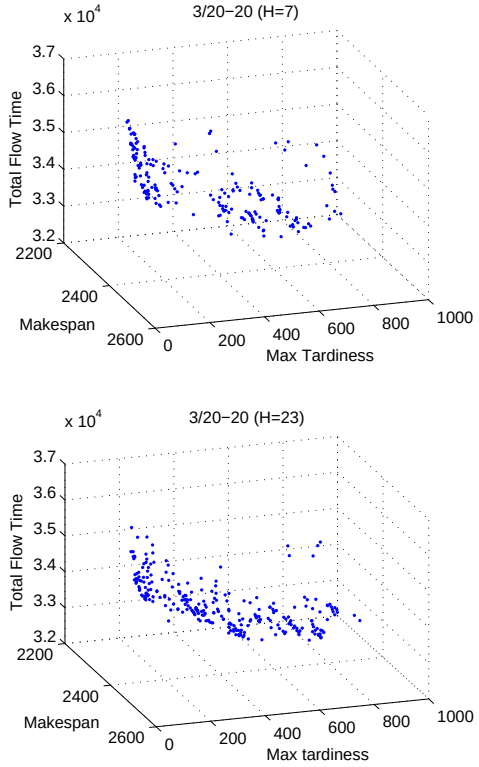


Fig. 8: Plot of non-dominated solutions with the lowest IGD-metric value found by MOEA/D-TS with $H = 7$ (top) and $H = 23$ (bottom) for 3-objective test instances with 20 jobs.

controlled by the user defined parameter H. In this section, we study the influence of H on the performance of MOEA/D-TS. We tested MOEA/D-TS with a smaller $H = 7$ (36 uniform weight vectors) and a larger $H = 23$ (300 uniform weight vectors) for 3-objective PFSP test problem with 20 jobs. Fig. 8 shows the distribution of non-dominated solutions found by MOEA/D-TS with $H = 7$ and $H = 23$. We can observe from this figure that MOEA/D-TS with the larger value of H performed better than that with the smaller values of H w.r.t diversity. Clearly, one can understand that more uniform weight vectors are needed in MOEA/D-TS when the size of PF is very large. However, a very large value of H will increase the computation time in MOEA/D-TS.

VI. Conclusions

In this paper a single-optimisation local search method, TS, has been integrated with the MOEA/D framework. At some points during the search process when search on single-objective subproblems get stuck in local optimal solutions, TS is used to escape from these solutions. The search history and TS strategy are utilised. We have used the multiobjective permutation flow shop scheduling problems as test problems. Our experimental results on the PFSP test instances indicated that our proposed MOEA/D-TS is a very effective technique that outperforms the classical MOEA/D. We have also shown that using a strategy for escaping Pareto local optimal solutions in MOEA/D is necessary for improving algorithm performance. As future work, the tuning parameters of the MOEA/D-TS will be investigated, and the MOEA/D-TS will be extended to handle other combinatorial optimisation problems.

References

[1] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[2] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[3] F. Glover, "Tabu search part I," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[4] C. Voudouris, "Guided local search for combinatorial optimisation problems." Ph.D. dissertation, University of Essex, 1997.

[5] L. Ke, Q. Zhang, and R. Battiti, "MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and ant colony," *IEEE Transactions on Cybernetics*, vol. PP, no. 99, pp. 1–15, 2013.

[6] H. Li and D. Landa-Silva, "An adaptive evolutionary multi-objective approach based on simulated annealing," *Evolutionary Computation*, vol. 19, no. 4, pp. 561–595, 2011.

[7] A. Alhindi and Q. Zhang, "MOEA/D with Guided Local Search: Some preliminary experimental results," in *the 5th Computer Science and Electronic Engineering Conference (CEEC)*, pp. 109–114, 2013.

[8] K. Miettinen, *Nonlinear multiobjective optimization*. MA, Norwell, USA: Kluwer Academic, 1999.

[9] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 204–223, 2003.

[10] Q. Zhang, W. Liu, and H. Li, "The performance of a new version of MOEA/D on CEC09 unconstrained mop test instances," in *CEC '09*, pp. 203–208, 2009.

[11] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flow-shop problem," *European Journal of Operational Research*, vol. 91, no. 1, pp. 160 – 175, 1996.

[12] F. Jin, S. Song, and C. Wu, "Structural property and meta-heuristic for the flow shop scheduling problem," in *Computational Intelligence in Flow Shop and Job Shop Scheduling*. Springer, pp. 1–20, 2009.

[13] J. Grabowski, J. Pempera, and Correspondence, "New block properties for the permutation flow shop problem with application in tabu search," *Journal of the Operational Res. Soc.*, vol. 52, no. 2, pp. 210–220, 2001.

[14] J. Grabowski and J. Pempera, "The permutation flow shop problem with blocking. a tabu search approach," *Omega*, vol. 35, no. 3, pp. 302 – 311, 2007.

[15] J. Grabowski and M. Wodecki, "A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion," *Comp. and Oper. Research*, vol. 31, no. 11, pp. 1891 – 1909, 2004.

[16] T. Murata and H. Ishibuchi, "Performance evaluation of genetic algorithms for flowshop scheduling problems," in *IEEE Conference on Evol. Comput.*, vol.2, pp. 812–817, 1994

[17] P. Soriano and M. Gendreau, "Diversification strategies in tabu search algorithms for the maximum clique problem," *Annals of Operations Research*, vol. 63, no. 2, pp. 189–207, 1996.

[18] F. Glover, M. Laguna, *Tabu Search*. Springer US, 1997.

[19] P. C. Chang, S.-H. Chen, Q. Zhang, and J.-L. Lin, "MOEA/D for flowshop scheduling problems," in *IEEE World Congress on Comput. Intelligence*, pp. 1433–1438, 2008.

[20] H. Ishibuchi, Y. Sakane, N. Tsukamoto, and Y. Nojima, "Adaptation of scalarizing functions in MOEA/D: An adaptive scalarizing function-based multiobjective evolutionary algorithm," in *Evolutionary Multi-Criterion Optimization*. Springer, pp. 438–452, 2009.

[21] H. Ishibuchi, N. Akedo, and Y. Nojima, "A study on the specification of a scalarizing function in MOEA/D for many-objective knapsack problems," in *Learning and Intelligent Optimization*. Springer, pp. 231–246, 2013.

[22] M. López-Ibánez, L. Paquete, and T. Stützle, "Exploratory analysis of stochastic local search algorithms in biobjective optimization," in *Experimental methods for the analysis of optimization algorithms*. Springer, pp. 209–222, 2010.