METAHEURISTICS FOR THE NO-IDLE PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Özge BÜYÜKDAĞLI

Supervisor: Mehmet Fatih TAŞGETİREN

Bornova, İZMİR

2013

YASAR UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

METAHEURISTICS FOR THE NO-IDLE PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Özge BÜYÜKDAĞLI

Thesis Advisor: Mehmet Fatih TAŞGETİREN

Department of Industrial Engineering

Bornova, İZMİR
2013

This study titled “Metaheuristics for the No-Idle Permutation Flowshop Scheduling Problem ” and presented as Master’s Thesis by Özge BÜYÜKDAĞLI has been evaluated in compliance with the relevant provisions of Y.U Graduate Education and Training Regulation and Y.U Institute of Science Education and Training Direction and jury members written below have decided for the defence of this thesis and it has been declared by consensus / majority of votes that the candidate has succeeded in thesis defence examination dated……..

Jury Members:	**Signature:**
Head:	…… ……...
Rapporteur Member:	……. ……
Member:	……. ……

TEXT OF OATH

I declare and honestly confirm that my study titled “Metaheuristics for the No-Idle Permutatıon Flowshop Scheduling Problem”, and presented as Master’s Thesis has been written without applying to any assistance inconsistent with scientific ethics and traditions and all sources I have benefited from are listed in bibliography and I have benefited from these sources by means of making references.

.. / .. / 20…

Özge BÜYÜKDAĞLI

Signature

ÖZET

BEKLEME ZAMANSIZ PERMÜTASYON AKIŞ TİPİ ÇİZELGELEME PROBLEMİ İÇİN SEZGİSEL YÖNTEMLER

BÜYÜKDAĞLI, Özge

Yüksek Lisans Tezi, Endüstri Mühendisliği Bölümü
Tez Danışmanı: Doç. Dr. Mehmet Fatih TAŞGETİREN

Mayıs 2013, 50 sayfa

Bu çalışmada, permütasyon akış tipi çizelgeleme probleminin, bekleme zamanlarına izin verilmeyen hali ele alınmıştır. Güçlü bir metasezgisel algoritma olan Genel Değişken Komşu Arama algoritması, dış döngüde ekle ve değiştir operasyonları, iç döngüde ise iteratif açgözlü algoritma ve iteratif bölgesel arama algoritması kullanılmıştır. Sunulan algoritmanın performansı, teknik yazında sunulan 4 farklı algoritmayla sonuçlarının karşılaştırılması ile ölçülmüştür. Karşılaştırma yapılan diğer algoritmalar şunlardır; (1) iteratif açgözlü, (2) değişken iteratif açgözlü, (3) hibrit ayrık farksal evrim algoritması, (4) farksal evrim ile değişken iteratif açgözlü algoritması. Bu algoritmaların performanslarını test etmek için http://soa.iti.es/rruiz sayfasında, Prof. Ruben Ruiz tarafından sunulan örnek problem yapısı kullanılmıştır. Yapılan karşılaştırmalar sonucunda Genel Değişken Komşu Arama algoritmasının, mevcut bilinen en iyi 250 sonucun 85 tanesini iyileştirdiği gözlenmiştir.

ABSTRACT

METAHEURISTICS FOR THE NO-IDLE PERMUTATION FLOWSHOP SCHEDULING PROBLEM

BÜYÜKDAĞLI, Özge

Master's Thesis, Department of Industrial Engineering
Supervisor: Assoc. Prof. Mehmet Fatih TAŞGETİREN

May 2013, 50 pages

In this thesis, a variant of permutation flowshop scheduling problem, where no-idle times are allowed on machines, is considered and a metaheuristic algorithm; a General Variable Neighborhood Search algorithm with insert and swap operations in outer loop and in the inner loop (Variable Neighborhood Descent phase), Iterated Greedy algorithm and Iterated Local Search algorithm is represented. The results of the algorithm are compared to the results with some other algorithms to measure the performance. These algorithms are; (1) an iterated greedy, (2) variable iterated greedy, (3) the hybrid discrete differential evolution and (4) variable iterated greedy algorithm with differential evolution algorithm. The performances of the proposed algorithms are tested on the Prof. Ruben Ruiz' benchmark suite that is presented in http://soa.iti.es/rruiz. Computational results are proposed and concluded as the GVNS algorithm further improved 85 out of 250 current best known solutions. In addition, these conclusions are supported by the paired T-tests and the interval plot.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my thesis advisor Assoc. Prof. Mehmet Fatih Taşgetiren for his invaluable support, encouragement and patience throughout my research.

I gratefully acknowledge Assist. Prof. Önder Bulut for his help and encouragement all the time.

This thesis would not have been possible without their encouragement and support.

I thankfully acknowledge the support of TUBITAK - Turkish Technological and Scientific Research Institute during my thesis period.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INDEX OF SYMBOLS AND ABBREVIATIONS

Symbols	**Explanations**
$F_m/\ prmu, no-idle/C_{max}$	m machine no-idle permutation flowshop problem with makespan minimization
j , n	jobs, total number of jobs
k , m	machines, total number of machines
$p(j,k)$	processing time of job j on machine m
S_k	starting time of processing jobs on machine k
π , π_j	job permutation, j^{th} job of permutation
$C(\pi_j, k)$	completion time of π_j on machine k
C_{max}	makespan
$O(n^x m)$	
PS	population size
F , CR	mutation scale factor, crossover probability

Abbreviations

PFSP	permutation flowshop problem
NIPFS	no-idle permutation flowshop
NEH	Nawaz, Enscore, Ham heuristic

IG	Iterated Greedy
VNS	Variable Neighborhood Search
GVNS	Global Variable Neighborhood Search
TFT	total flow time
IG_LS	iterated greedy with local search
VIG_FL	variable iterated greedy by Framinan and Leisten
DE	differential evolution
DDE	discrete differential evolution
HDDE	hibrit discrete differential evolution
VIG_DE	variable iterated greedy with differential evolution
VND	variable neighborhood descent
ILS	iterated local search
RIS	referenced insertion
NFL	No-Free Lunch

CHAPTER 1

INTRODUCTION

A flowshop is a commonly used production system in manufacturing industries. Generally, in manufacturing environments, the jobs should go through different processes till the end items are obtained. If the route of each job is different, then this environment is referred as jobshop. The production environment with all jobs have the same route is called flowshop. Scheduling of a flowshop has an essential role in competitive environments; therefore this problem has been one of the most attractive subjects for researchers.

In a flowshop, there is more than one machine and each job must be processed on each of the machines. Each job has the same ordering of machines for its process sequence. Each job can be processed on one machine at a time, and each machine can process only one job at a time. For the permutation flowshop, the processing sequences of the jobs are the same on each machine. In other words, jobs have a permutation and therefore, once a permutation is fixed for all jobs on the first machine, this permutation is maintained for all other machines. If one job is at the i^{th} position on machine 1, then this job will be at the i^{th} position on all the machines.

In order to measure the performance of scheduling in a flowshop, there are several criteria such as, makespan and due-date based performance measures. Makespan criterion, without any doubt, the most widely used performance measure in the literature. The popularity of makespan criterion comes from the ease of implementation of this criterion to each kind of problem. On the other hand, in real life considerations, meeting customers' requirements on-time is essential for several industries. For these problems which aim to satisfy promised due dates to customers, due-date based performance measures have been attracted interest from the researchers recently. There are many interesting and successful studies that are considering total tardiness criterion in literature as well.

In this thesis, a variant of permutation flowshop scheduling problem (PFSP), where no-idle times are allowed on machines, is considered. The no-idle constraint has an important role in scheduling environment, where expensive

machinery is employed. Idling machines in such environments is not cost-effective. Another situation that production environment desires to have no-idle times in schedule, is when high setup time or costs exist so that shutting down the machines after initial setup is not wanted. In no-idle permutation flowshop scheduling (NIPFS) problem, each machine must process each job without any interruption from the beginning of the first job to the completion of the last job. In order to meet this constraint, delays may occur in the processing of the first job on any machine. There are various examples of this problem in manufacturing industries. For instance, fiberglass processing has both costly and time consuming setups where the furnaces must be heated up to 2800ºF which takes three days. So furnace must stay on during the entire production to avoid long setup times but at the same time, since the cost of idling furnace is too high, the production must be scheduled considering no-idle restriction. (H. Saadani, A. Guinet, M. Moalla, 2003) presented a three-machine flowshop production of engine blocks in a foundry.

$F_m/\text{prmu}, \text{no}-\text{idle}/C_{\max}$ is a well-known notation of the m-machine NIPFS problem where the makespan is minimized. (Baptiste, P., & Lee, K. H., 1997) showed that $F_3/\text{prmu}, \text{no}-\text{idle}/C_{\max}$ is an NP-hard problem. Although it has a great importance in both theory and practical applications, it has not attracted much attention in the literature by the researchers. In (Adiri, I. and Pohoryles, D., 1982) an algorithm to solve $F_2/\text{prmu}, \text{no}-\text{idle}/C_{\max}$ optimally, is presented. The first time, problem is studied with the makespan criterion in (Vachajitpan, 1982). (Woollam, 1986) examined heuristic approaches for the general m-machine no-idle PFSP with the makespan criterion.

Recently, heuristic approaches have attracted increasing attention by many researchers. The solution quality of heuristic approaches started to get higher especially when the low computational effort is considered. A heuristic, based on the traveling salesman problem (TSP), for the the $F_3/\text{prmu}, \text{no}-\text{idle}/C_{\max}$ was represented in (Saadani, N. E. H., Guinet, A., and Moalla, M., 2005). (Kalczynski, P.J. and Kamburowski, J., 2007) presented an adaptation of the NEH heuristic for the NIPFS problem and also studied the interactions between the no-idle and no-wait flowshops. In (Ruiz, R. , Vallada, E. , Fernández-Martínez, C., 2009), an IG algorithm for the NIPFS problem with the makespan criterion was presented and examined the performance against the existing algorithms. (Tasgetiren M.F., Pan

Q., Suganthan P.N.,Oner A.) presented a discrete artificial bee colony algorithm to solve the no-idle permutation flowshop scheduling problem with the total tardiness criterion. (Kirlik G., Oguz C., 2012) applied a different algorithm to a different problem; the single machine scheduling problem to minimize the total weighted tardiness with the sequence dependent setup times by using general variable neighborhood search (GVNS) algorithm. This algorithm resulted very well for that NP-hard problem, therefore, inspiring from (Kirlik G., Oguz C., 2012), in this study, a GVNS algorithm is proposed to solve the NIPFS problem with makespan criterion and compared the results with some other algorithms to measure the performance.

This paper is organized as follows. In Chapter 2 NIPFS problem is defined. Details of metaheuristic algorithms are given in Chapter 3. Computational experiments that evaluate the performance of the solution methods are reported in Chapter 4. Finally, conclusion is given in Chapter 5.

CHAPTER 2

NO-IDLE PERMUTATION FLOWSHOP SCHEDULING PROBLEM

No-idle permutation flowshop scheduling is required when the production environment desires to have no-idle times in production schedule because of the high costs or setup complexity of the system. In order to avoid the troubles in the production environment, the schedule must be done carefully while considering the all systems behavior.

There are n $(j = 1,2,..,n)$ jobs to be processed successively on m $(k = 1,2,..,m)$ machines with the same sequence on each machine. Associated with each job j and machine k, there is a processing time $p(j,k)$.

The assumptions for this problem are introduced;

- Each machine can perform at most one job at any given time;
- Each job can be processed on at most one machine at any given time;
- Processing sequences of jobs are same on each machine;
- There cannot be idle times between the start of processing the first job to the completion of processing the last job on any machine.

While constructing the algorithm that will determine the production sequence of jobs, the time complexity of performance measures must be considered as well as problems' objective, which is the makespan minimization for this problem. Both the reliability and the speed of the algorithm have an essential importance for real life problems. In real life problems, it is desirable to have a good quality solution in a short time period. In order to provide this, researchers have been studying on decreasing the time complexity of algorithms. (Ruiz, R. , Vallada, E. , Fernández-Martínez, C., 2009) proposed a formulation to calculate makespan of no-idle flowshop. In this formulation, first, *when a given machine can start processing with no needed idle time* is calculated. Then by using these values, the completion times are calculated straight forward, by adding processing times of jobs to the starting times for each machine, since the jobs are processed with no-idle time. Moreover, (Pan, Q-K. and Wang, L., 2008) also presented a formulation for the NIPFS problem with the makespan criterion.

This formulation consists of forward and backward pass calculation. These methods decrease the time complexity of calculating the completion times comparing to (Ruiz, R. , Vallada, E. , Fernández-Martínez, C., 2009) calculation method, which is a very desirable property especially in algorithmic studies. Less CPU time for calculation, provide opportunity to algorithm to apply more moves or to generate more generations that increases the chance to obtain more improved solutions.

In the formulation that (Ruiz, R. , Vallada, E. , Fernández-Martínez, C., 2009) used in their study, the necessity of the delay of the jobs, to ensure that jobs are processed without idle time, is considered and according to this feature, they proposed to first calculate the starting time of processing jobs on each machine which is denoted by $S_k, k = \{1, .., m\}$ where $S_1 = 0$ and k denotes the machine. Then by adding the processing times to these starting times, they calculate the completion times. Let, a job permutation, $\pi = \{\pi_1, \pi_2, .., \pi_n\}$ be the sequence of jobs to be processed on each machine and the completion time of $\pi_j$on machine k be $C(\pi_j, k)$. The formulations are given below;

$$S_k = S_{k-1} + \max_{1 \le h \le n}\left\{\sum_{j=1}^{h} p(\pi_j, k-1) - \sum_{j=1}^{h-1} p(\pi_j, k)\right\},$$

$$k = \{2, .., m\} \quad (1)$$

After calculating the S_k values, the completion time of each job can be calculated by adding the processing times of jobs on each machine;

$$C(\pi_1, k) = S_k + p(\pi_1, k) \qquad k = \{1, .., m\} \quad (2)$$

$$C(\pi_j, k) = C(\pi_{j-1}, k) + p(\pi_j, k) \qquad j = \{2, .., n\}, k = \{1, .., m\} \quad (3)$$

As a result, the completion time of jobs on the last machine gives the makespan;

$$C_{max} = C(\pi_n, m). \quad (4)$$

In order to come up with the complexity, the summations inside the max term in expression (1) have to be stored at each step. For example:

$$\sum_{j=1}^{h} p(\pi_j, k-1) = \sum_{j=1}^{h-1} p(\pi_j, k-1) + p(\pi_h, k-1) \quad (5)$$

As mentioned before, (Pan, Q-K. and Wang, L., 2008) proposed another calculation method for NIPFS problem that consists of forward and backward passes which is explained in the following sections.

2.1 Forward Pass Calculation

Let the partial sequence of π, $\pi_j^E = \{\pi_1, \pi_2, .., \pi_j\}$ represent the sequence of jobs from the first job to the j^{th} job of sequence π where $1 < j < n$. The minimum difference, between the completion of processing the last job of π_j^E on machines k and $k+1$ is denoted as $F(\pi_j^E, k, k+1)$ and restricted by no-idle constraint. $F(\pi_j^E, k, k+1)$ can be computed as shown below.

$$F(\pi_1^E, k, k+1) = p(\pi_1, k+1) \qquad k = 1,2,..,m-1 \qquad (6)$$

$$F(\pi_j^E, k, k+1) = \max\{F(\pi_{j-1}^E, k, k+1) - p(\pi_j, k), 0\} + p(\pi_j, k+1)$$
$$j = 2,3,..,n \quad and \quad k = 1,2,..,m-1 \qquad (7)$$

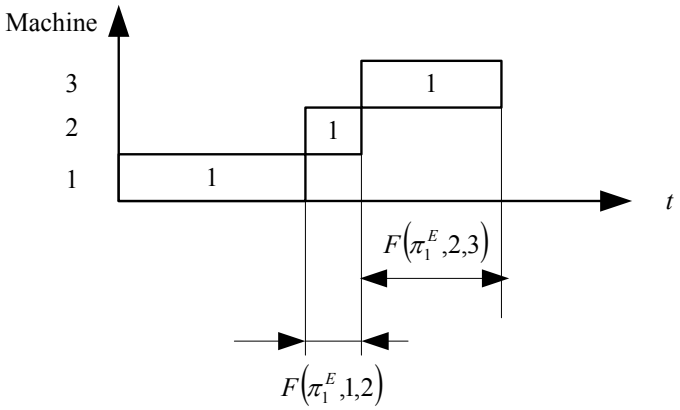


Figure 1. Computation of $F(\pi_1^E, k, k+1)$

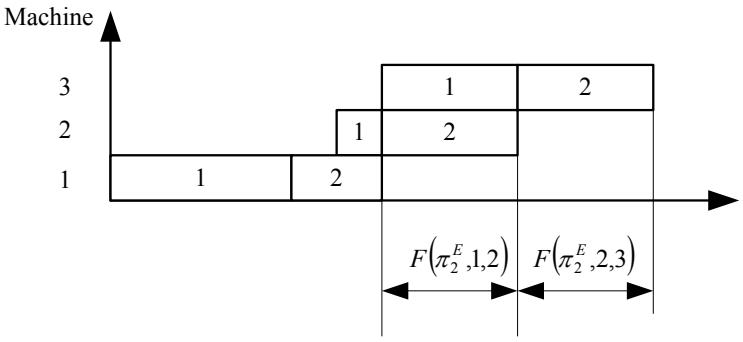


Figure 2. Computation of $F(\pi_2^E, k, k+1)$

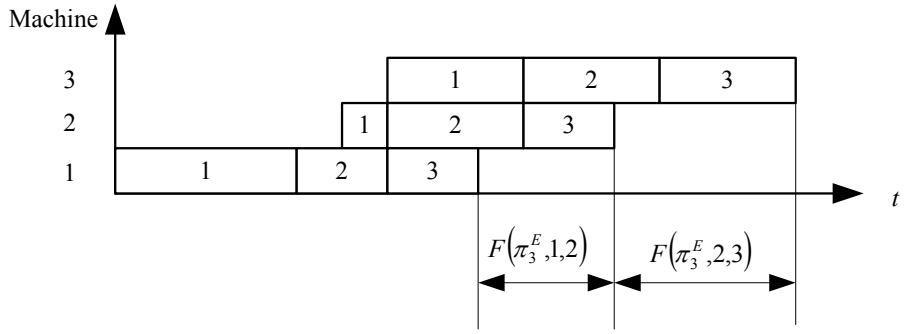


Figure 3. Computation of $\boldsymbol{F(\pi_3^E, k, k+1)}$

In formulation (6), difference between the completions of processing the last job of π_1^E which only includes one job, on machines k and $k+1$ is given. Since there is only one job, $F(\pi_1^E, k, k+1)$ can be calculated by considering processing time of that job on corresponding, $(k+1)^{th}$ machine. In formulation (7), calculation of $F(\pi_j^E, k, k+1)$ for $j > 1$ is represented. It can be calculated by not only considering processing time of j[th] job on machine k, also adding the positive difference between the previous job's completion of processing on machines k and $k+1$.

The completion time of last job, π_n on last machine m can be calculated as summation of $F(\pi_n^E, k, k+1)$ value for all machines and the processing times of all previously processed jobs including π_n itself;

$$C(\pi_n, m) = C_{max}(\pi_n^E) = \sum_{k=1}^{m-1} F(\pi_n^E, k, k+1) + \sum_{j=1}^{n} p(\pi_j, 1) \quad (8)$$

Then, for any job j, completion time on last machine m can be computed by subtracting the processing time of the next job, π_{j+1} from the completion time of π_{j+1}on machine m;

$$C(\pi_j, m) = C(\pi_{j+1}, m) - p(\pi_{j+1}, m) \qquad j = n-1, n-2, .., 1 \quad (9)$$

Makespan can also be defined as the maximum completion time of jobs on the last machine by using the no-idle constraint of this problem;

$$C_{max}(\pi_n^E) = \max(C(\pi_1, m), C(\pi_2, m), .., C(\pi_n, m)) \quad (10)$$

And, the total flow time of the permutation π, can be obtained as a summation of all completion times;

$$TFT(\pi) = \sum_{j=1}^{n} C(\pi_j, m) \quad (11)$$

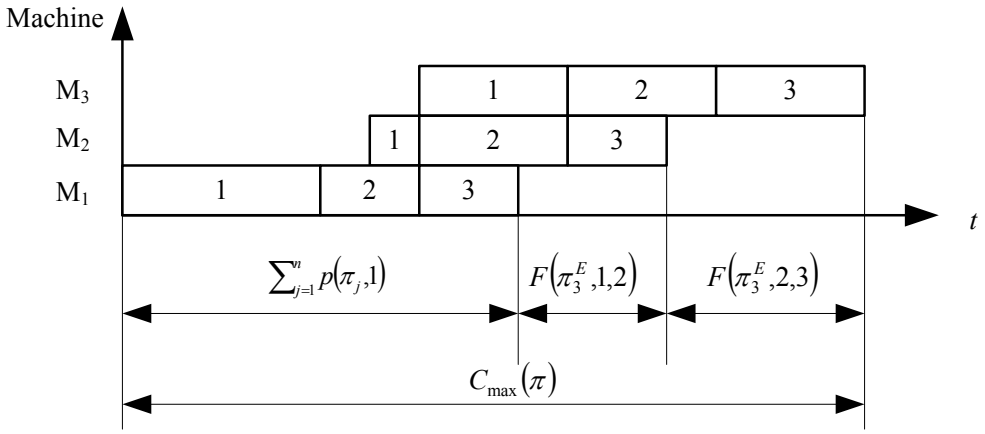


Figure 4. Computation of $\mathbf{C_{max}(\pi)}$

For the example instance for 3-job 3-machine problem that is taken from (Tasgetiren M.F., Pan Q., Suganthan P.N.,Oner A., In press), Figure 1 a to Figure 1 d, the forward calculation is illustrated with a permutation $\pi = \{1,2,3\}$.

An example for forward pass calculation is represented below.

Example

An example instance is given in Table 1 for 3-job 3-machine problem with permutation $\pi = \{1,2,3\}$ and with due date tightness factor of $\tau = 1$. According to the data given, the forward pass calculation is presented below, in detail.

	Machines (k)		
Jobs (j)	1	2	3
p_{1j}	4	1	3
p_{2j}	2	3	3
p_{3j}	2	2	3

Table 1. An example instance for forward and backward pass calculation

By using the equation (6), $F\left(\pi_j^E, k, k+1\right)$ for the first job is computed as;

$F(\pi_1^E, 1,2) = p(1,2) = 1$

$F(\pi_1^E, 2,3) = p(1,3) = 3$

Equation (7) is used to calculate $F\left(\pi_j^E, k, k+1\right)$ for the remaining jobs.

For $j = 2$ and $= 1,2$;

$$F(\pi_2^E, 1,2) = \max\{F(\pi_1^E, 1,2) - p(\pi_2, 1), 0\} + p(\pi_2, 2)$$
$$= \max\{F(\pi_1^E, 1,2) - p(2,1), 0\} + p(2,2)$$
$$= \max\{(1-2), 0\} + 3 = 3$$
$$F(\pi_2^E, 2,3) = \max\{F(\pi_1^E, 2,3) - p(\pi_2, 2), 0\} + p(\pi_2, 3)$$
$$= \max\{F(\pi_1^E, 2,3) - p(2,2), 0\} + p(2,3)$$
$$= \max\{(3-3), 0\} + 3 = 3$$

For $j = 3$ and $= 1,2$;

$$F(\pi_3^E, 1,2) = \max\{F(\pi_2^E, 1,2) - p(\pi_3, 1), 0\} + p(\pi_3, 2)$$
$$= \max\{F(\pi_2^E, 1,2) - p(3,1), 0\} + p(3,2)$$
$$= \max\{(3-2), 0\} + 2 = 3$$
$$F(\pi_3^E, 2,3) = \max\{F(\pi_2^E, 2,3) - p(\pi_3, 2), 0\} + p(\pi_3, 3)$$
$$= \max\{F(\pi_2^E, 2,3) - p(3,2), 0\} + p(3,3)$$
$$= \max\{(3-2), 0\} + 3 = 4$$

The completion time of the last job $j = 3$ on last machine $k = 3$ which gives also the makespan of the permutation is computed by using the equation (8);

$$C(\pi_3, 3) = \sum_{k=1}^{3-1} F(\pi_3^E, k, k+1) + \sum_{j=1}^{3} p\left(\pi_j, 1\right)$$
$$= F(\pi_3^E, 1,2) + F(\pi_3^E, 2,3) + p(1,1) + p(2,1) + p(3,1)$$
$$= 3 + 4 + 4 + 2 + 2 = 15$$

For remaining jobs, $j = 1,2$ completion time on last machine $m = 3$ is computed by using equation (9);

$$C(\pi_2, 3) = C(\pi_3, 3) - p(\pi_3, 3)$$
$$= 15 - 3 = 12$$
$$C(\pi_1, 3) = C(\pi_2, 3) - p(\pi_2, 3)$$
$$= 12 - 3 = 9$$

The makespan can also be computed by using (10);

$$C_{max} = \max(C(\pi_1, 3), C(\pi_2, 3), C(\pi_3, 3))$$
$$= \max(15,12,9)$$
$$= 15$$

As mentioned before, the makespan calculation by using equation (8) and equation (10) results same since the problem has a no-idle constraint.

Total flow time is calculated with equation (11);

$$TFT(\pi) = \sum_{j=1}^{3} C(\pi_j, 3)$$
$$= C(\pi_1, 3) + C(\pi_2, 3) + C(\pi_3, 3)$$
$$= 36$$

2.2 Backward Pass Calculation

Let, the partial sequence of π, $\pi_j^F = \{\pi_j, \pi_{j+1}, .., \pi_n\}$ represent the sequence of jobs from the j^{th} job to the last job n of sequence π where $1 < j < n$. And let $E(\pi_j^F, k, k+1)$ be the lower bound for the minimum difference between the start of processing the first job of π_j^F on machines k and $k+1$. Then;

$$E(\pi_n^F, k, k+1) = p(\pi_n, k) \qquad k = 1,2,..,m-1 \qquad (12)$$

$$E(\pi_j^F, k, k+1) = \max\{E(\pi_{j+1}^F, k, k+1) - p(\pi_j, k+1), 0\} + p(\pi_j, k)$$
$$j = n-1, n-2, .., 1 \quad and \quad k = 1,2,..,m-1 \qquad (13)$$

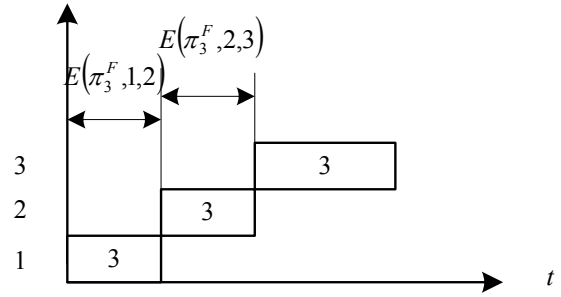


Figure 5. Computation of $\mathbf{E}(\boldsymbol{\pi}_3^F, \mathbf{k}, \mathbf{k}+\mathbf{1})$

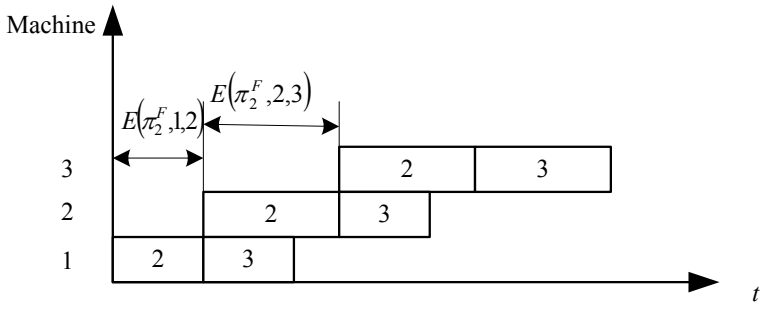


Figure 6. Computation of $\boldsymbol{E}(\boldsymbol{\pi}_2^F, \boldsymbol{k}, \boldsymbol{k}+\mathbf{1})$

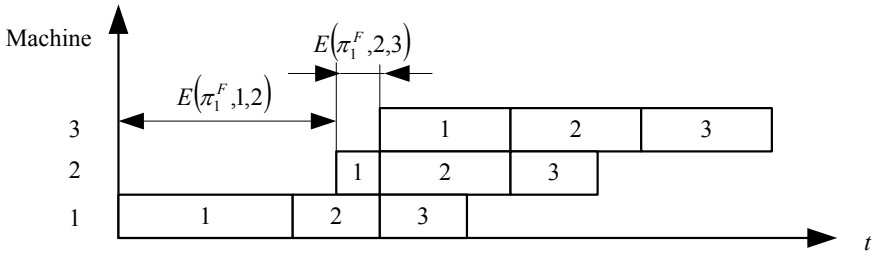


Figure 7. Computation of $\mathbf{E}(\boldsymbol{\pi_1^F}, \mathbf{k}, \mathbf{k+1})$

The completion time of the first job π_1 on last machine m can be obtained as follows;

$$C(\pi_1, m) = \sum_{k=1}^{m-1} E(\pi_1^F, k, k+1) + p(\pi_1, m) \quad (14)$$

Then, the completion time of any job π_{j+1} on the last machine m can be computed as;

$$C(\pi_{j+1}, m) = C(\pi_j, m) + p(\pi_{j+1}, m) \qquad j = 1,2,..,n-1 \quad (15)$$

The objective of no-idle permutation flowshop is to find the permutation π^* which has a minimum makespan or total flow time in the set of all permutations $\prod$. The permutation π^* can be obtained as;

$$C_{max}(\pi^*) \leq C_{max}(\pi_n^E) \text{ or } C_{max}(\pi^*) \leq C_{max}(\pi_1^F)\ , \forall \pi \in \prod \quad (16)$$

$$TFT(\pi^*) \leq TFT(\pi_n^E) \text{ or } (\pi^*) \leq TFT(\pi_1^F)\ , \forall \pi \in \prod \quad (17)$$

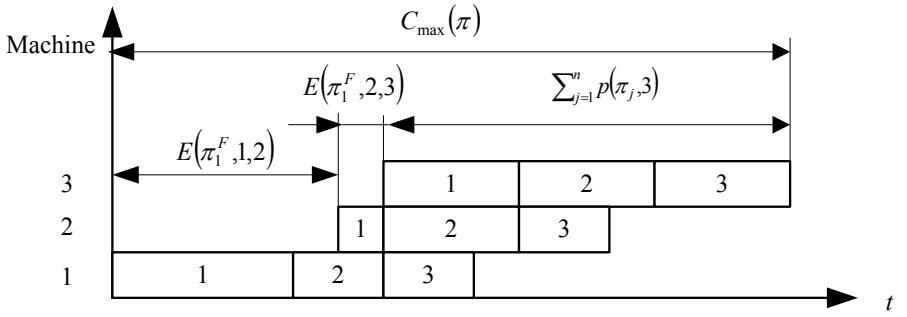


Figure 8. Computation of $\mathbf{C_{max}(\pi)}$

Fig. 2 a to Fig.2 d illustrate the backward pass calculation of makespan for a 3-job 3-machine problem.

By using the example instance that is given in Table 1, a detailed backward pass calculation is represented below.

Example

For the last job $j = 3$, $E(\pi_j^F, k, k+1)$ is computed by using equation (12);

$E(\pi_3^F, 1,2) = p(\pi_3, 1) = p(3,1) = 2$

$E(\pi_3^F, 2,3) = p(\pi_3, 2) = p(3,2) = 2$

Equation (13) is used to calculate $E(\pi_j^F, k, k+1)$ for the remaining jobs.

For $j = 2$ and $= 1,2$;

$$E(\pi_2^F, 1,2) = \max\{E(\pi_3^F, 1,2) - p(\pi_2, 2), 0\} + p(\pi_2, 1)$$
$$= \max\{E(\pi_3^F, 1,2) - p(2,2), 0\} + p(2,1)$$
$$= \max\{(2-3), 0\} + 2 = 2$$

$$E(\pi_2^F, 2,3) = \max\{E(\pi_3^F, 2,3) - p(\pi_2, 3), 0\} + p(\pi_2, 2)$$
$$= \max\{E(\pi_3^F, 2,3) - p(2,3), 0\} + p(2,2)$$
$$= \max\{(2-3), 0\} + 3 = 3$$

For $j = 1$ and $= 1,2$;

$$E(\pi_1^F, 1,2) = \max\{E(\pi_2^F, 1,2) - p(\pi_1, 2), 0\} + p(\pi_1, 1)$$
$$= \max\{E(\pi_2^F, 1,2) - p(1,2), 0\} + p(1,1)$$
$$= \max\{(2-1), 0\} + 4 = 5$$

$$E(\pi_1^F, 2,3) = \max\{E(\pi_2^F, 2,3) - p(\pi_1, 3), 0\} + p(\pi_1, 2)$$
$$= \max\{E(\pi_2^F, 2,3) - p(1,3), 0\} + p(1,2)$$
$$= \max\{(3-3), 0\} + 1 = 1$$

The completion time for the first job $j = 1$ on the last machine $k = 3$, is obtained by equation (14);

$$C(\pi_1, 3) = \sum_{k=1}^{2} E(\pi_1^F, k, k+1) + p(\pi_1, 3)$$
$$= E(\pi_1^F, 1,2) + E(\pi_1^F, 2,3) + p(\pi_1, 3)$$
$$= 5 + 1 + 3 = 9$$

Using equation (15), all other remaining jobs' $j = 2,3$ completion time on last machine $m = 3$ is computed;

$$C(\pi_2, 3) = C(\pi_1, 3) + p(2,3)$$
$$= 9 + 3 = 12$$

$$C(\pi_3, 3) = C(\pi_2, 3) - p(3,3)$$
$$= 12 + 3 = 15$$

As mentioned before, the makespan can also be computed by using (10);

$$C_{max} = \max(C(\pi_1, 3), C(\pi_2, 3), C(\pi_3, 3))$$
$$= \max(9,12,15)$$
$$= 15$$

And the total flow time can be calculated same as forward pass method by using equation (11).

As a result of this comparison study, the formulation of (Pan, Q-K. and Wang, L., 2008) is selected to use in this study.

CHAPTER 3

METAHEURISTIC ALGORITHMS

In the last years, a new kind of approximate algorithm started to be used by many researchers, which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are called metaheuristics.

Metaheuristic algorithms basically aim to provide nearly-optimal solutions by creating an initial solution and improving this solution iteratively. These algorithms help researchers to have qualified solutions for very complex problems (i.e. NP-Hard problems) which are commonly needed to be solved in real-life cases. There are several algorithms presented in literature; some of these algorithms uniquely developed for a specific problem, some of these are applicable for different types of problems. (Stützle, 1999) presented the definition for the term metaheuristic as follows;

"Metaheuristics are typically high-level strategies which guide an underlying, more problem pecific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descents by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form."

In literature, for NIPFS problem, many researchers proposed different algorithms. In this thesis, the results obtained from four different algorithms that are proposed for the NIPFS problem, are compared with the proposed GVNS algorithm. These algorithms are;

- An Iterated Greedy (IG_LS) algorithm for the NIPFS problem with the makespan criterion that is presented in (Ruiz, R. , Vallada, E. , Fernández-Martínez, C., 2009)

- Variable Iterated Greedy (VIG_FL) algorithm, that is presented and implemented to solve the PFSP with the total tardiness criterion in (Framinan and Leisten, 2008)
- The Hybrid Discrete Differential Evolution (HDDE) algorithm that is proposed by (Deng G and Gu X., 2012)
- Variable Iterated Greedy algorithm with Differential Evolution (VIG_DE).

In this chapter these algorithms and the algorithm that is proposed; a GVNS algorithm with insert and swap operations in outer loop and in the inner loop (VND), IG algorithm and iterated local search (ILS) algorithm are explained.

3.1 Iterated Greedy (IG) Algorithm

Iterated Greedy algorithm is presented in (Ruiz R., Stützle T., 2007), based on a very simple principle and easy to implement as well as has successful applications in discrete/combinatorial optimization problem. It produces solutions with very good quality in a very short amount of time.

Algorithm starts with an initial solution that is generated either randomly or using heuristics, such as NEH heuristic that is explained in Section 3.1.1 . Then Destruction Construction Procedure is applied which is presented in Section 3.1.2 and repeated until some stopping criterion like a maximum number of iterations or a computation time limit is met. An optional local search phase can be added before the acceptance test for improving the re-constructed solution. For this study, Referenced Insertion Algorithm is used as a local search and explained in Section 3.1.3. . The pseudo code of the algorithm is given below;

$Procedure\ IG(\pi, d)$
$\pi = \pi_b$
$\pi = LocalSearch(\pi)$
$do\{$
$\pi_1 = DestructConstruct(\pi, d)$
$\pi_2 = RIS(\pi_1)$ //(optional) Local Search
$if\ f(\pi_2) < f(\pi)\ then$ //Acceptance test
$\pi = \pi_2$
$\pi^R = \pi_2$
$if\ f(\pi_2) < f(\pi_b)\ then$
$\pi_b = \pi_2$
$endif$
$elseif\ random \leq exp\{-(f(\pi_2) - f(\pi))/T\}$

$\pi = \pi_2$
$\pi^R = \pi_2$
$endif$
$\}while\ (NotTermination)$ //Stopping criterion
$return\ \pi\ and\ \pi_b$
$endprocedure$

Figure 9 Iterated Greedy Algorithm

3.1.1 The NEH Heuristic

NEH heuristic is proposed by (Nawaz, M., Enscore, Jr, E. E., and Ham, I., 1983) and has been recognized as the highest performing method for the permutation flowshop scheduling problem. The NEH algorithm is based on scheduling jobs with high processing times, as early as possible, on all the machines. The NEH heuristic has three phases:

1. For each job j, the total processing time on the m machines are computed: $P_j = \sum_{k=1}^{m} p_{jk}\ , \forall j$
2. Jobs are sorted in descending order of $P_j\ s$. Let the resulting permutation be π, then the first two jobs are selected and two possible permutations are generated and the one that results with the minimum makespan or total flowtime is selected.
3. Second phase is repeated until all jobs are sequenced. In order to generalize the procedure; in the i^{th} step, the job π_i at position i is taken and inserted into i possible positions of the permutation of the jobs that are already scheduled. The best resulting permutation is selected.

The computational complexity of the NEH heuristic is $O(n^3m)$, which consumes a very high level of CPU especially for larger instances. There are some speed-up methods that are introduced to reduce the complexity of NEH to $O(n^2m)$. It is claimed that these speed-up methods are one of the key factors to the success of most algorithms especially the ones with the makespan criterion. Even though, in this study no speed-up method is used, the result of the proposed algorithm is way better than most of the algorithms from the literature.

3.1.2 Destruction and Construction Procedure

Destruction and Construction Procedure consists of two main steps; destruction step and construction step. In the destruction step, pre-determined

parameter d many jobs are randomly chosen and removed from the current solution. Therefore, two partial solutions obtained; one consists of the removed jobs, in the order which they removed denoted as π^R, the other one is the remaining part of the initial solution with size $n-d$ and denoted as π^D. In the construction phase, a heuristic called NEH insertion is used. In this heuristic, basically all jobs in π^R is inserted into each position in π^D one by one, and finally the best permutation with the minimum makespan (or total tardiness) is selected. In a more detailed way; the first job of the π^R is selected and removed from π^R and inserted into all possible $n-d+1$ positions, thus $n-d+1$ many partial solutions obtained. By considering the performance criterion, the best solution is selected and kept. Next, the same procedure is applied for second job, third job and so on, until the π^R is empty. Therefore, the size of π^D becomes n again. In order to be more descriptive, an example that is represented in (Ruiz R., Stützle T., 2007) is given below.

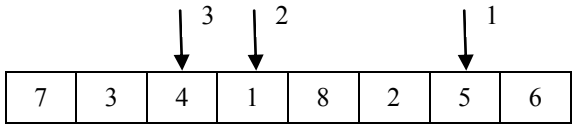
Machines (i)	Jobs (*j*)							
	1	2	3	4	5	6	7	8
1	456	654	852	145	632	425	214	654
2	789	123	369	678	581	396	123	789
3	654	123	632	965	475	325	456	654
4	321	456	581	421	32	147	789	123
5	456	789	472	365	536	852	654	123
6	789	654	586	824	325	12	321	456
7	654	321	320	758	863	452	456	789
8	789	147	120	639	21	863	789	654

Table 2. An example instance for Destruction and Construction Procedure

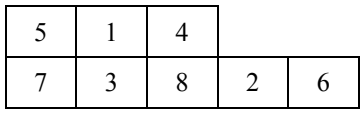
For the example instance given in Table 2, one iteration of the IG algorithm is applied. Let the given sequence given below be the initial solution that is obtained by using NEH algorithm with $C_{max} = 8564$.

7	3	4	1	8	2	5	6

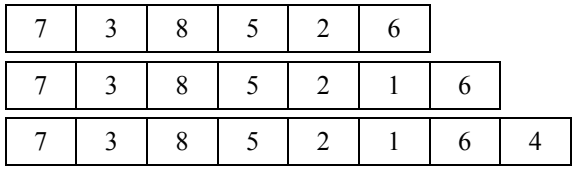
For the destruction phase, let the destruction size be, $d = 3$. Then, three jobs must be randomly chosen and removed from the current solution. Let these jobs be 5, 1 and 4, respectively.



Then, the removed job to be reinserted, π^R and the partial sequence to be reconstructed π^D becomes as follows, respectively;



In construction phase, each job in π^R is reinserted in all possible positions in π^D and the sequence with the best performance is selected. In the figures below, the best sequence after the reinsertion of each job is given.



After reinserting job 5, $C_{max} = 7589$

After reinserting job 1, $C_{max} = 8243$

After reinserting job 4, $C_{max} = 8366$

The example shows that a new solution obtained by removing jobs 5, 1 and 4 and reinserting them, is $C_{max} = 8366$ which is a better solution than the inital solution given by the NEH heuristic ($C_{max} = 8564$). This new solution is accepted by the acceptance criterion since it is better than the starting solution. Furthermore, this new sequence is known to be an optimal solution for this instance.

3.1.3 Referenced Insertion Algorithm

In the referenced insertion (RIS) procedure, as an initial step, a referenced permutation, π^R, is selected which is the best solution found so far. Then, the first job of the π^R is determined and the position of this job is found in the current permutation π. This corresponding job is removed from π and inserted into all possible positions of permutation π. Next, second job of the π^R is found in the permutation π, removed and inserted into the positions of its own permutation. And the procedure goes on in this way, until all the jobs in the π^R is processed. For example, let the referenced sequence be, $\pi^R = \{4,2,5,1,3\}$ and the current

solution be $\pi = \{2,1,5,4,3\}$. The RIS procedure selects the first job of π^R, which is job 4 and finds it in the current sequence. It removes job 4 from π and inserts into all possible positions in π. The objective function values of these newly generated permutations are compared to π and if any insertion is better than π, then the current solution is replaced by that permutation obtained by the insertion. Then the RIS procedure takes the second job of the referenced sequence which is job 2 and finds it in π. Procedure removes job 2 from current solution π, and inserts it into all possible positions of π. This procedure is repeated until π^R is empty.

The RIS procedure claims to have better solutions since the jobs are selected by referring to a good quality solution instead of a random choice. The pseudo code of this algorithm is represented below;

$Procedure\ RIS(\pi, \pi^R)$
$h = 1$
$i = 1$
$while(i \leq n)do\ \{$
    $h = h(mod)n$
    $\pi_1 = remove\ job\ \pi_k\ from\ \pi, corresponding\ to\ job\ \pi_h^R\ in\ the\ reference\ permutation\ \pi^R$
    $\pi_2 = the\ best\ permutation\ obtained\ by\ inserting\ job\ \pi_k\ in\ any\ possible\ position\ of\ \pi_1$
    $if\big(f(\pi_2) < f(\pi)\big)then\ \{$
        $\pi = \pi_2$
        $i = 1$
    $\}\ else\ \{$
        $h = h + 1$
    $\}endif$
$\}endwhile$
$return\ \pi$
$endprocedure$

Figure 10. Referenced Insertion Algorithm

In this study, an IG algorithm with RIS local search is applied to NIPFS problem. In addition to IG, another powerful algorithm, Iterated Local Search (ILS) algorithm is also applied to this problem. In the next section, ILS algorithm is explained in details.

3.1.4 Iterated Local Search Algorithm

Iterated local search (ILS) is first presented in (H.R. Lourenc, O. Martin, T. Stützle, 2002) and known as a simple and powerful stochastic local search method. According to (H.R. Lourenc, O. Martin, T. Stützle, 2002), "*ILS is a simple and generally applicable stochastic local search method that iteratively applies local search to perturbations of the current search point, leading to a randomized walk in the space of local optima"*. The main idea of iterated local search (ILS) algorithm is to apply local search repeatedly to initial solutions obtained by perturbations of a previously visited locally optimal solutions. The simplicity, ease of implementation and at the same time efficiency of this algorithm makes this algorithm eligible.

In (Thomas Stützle, 2006), the application of ILS to the quadratic assignment problem is represented. In this study, as a second algorithm of VND phase of the GVNS algorithm is inspired by this application of ILS.

In ILS algorithm, there are some procedures to be specified. These are;

- How to generate initial solution (*GenerateInitialSolution*),
- Perturbation type (*Perturbation*),
- Acceptance criterion (*AcceptanceCriterion*),
- Which local search to use (*LocalSearch*).

(Thomas Stützle, 2006), used a random assignment of items to locations as the initial solution (*GenerateInitialSolution*). The phase *Perturbation* exchanges k randomly chosen items, corresponding to a random move in the k-opt neighborhood. (Thomas Stützle, 2006) decided to determine value k by using VNS. In order to decide which solution to choose, as *AcceptanceCriterion, Better(s,s')* function is used. By using this function, good solutions (s) are determined as;

$$s = Better(s, s') = \begin{cases} s' & if \quad f(s') < f(s) \\ s & otherwise \end{cases}$$

where $f(s)$ represents the objective function value for solution s. In *LocalSearch* phase, an iterated descent algorithm with a first improvement pivoting rule is used. The pseudo code of the ILS algorithm is given below, in Figure 17.

$Procedure\ Iterated\ Local\ Search$

$\pi = GenerateInitialSolution$
$\pi_1 = LocalSearch(\pi)$
$do\{$
$\pi_2 = Perturbation(\pi_1, history)$
$\pi_3 = LocalSearch(\pi_2)$
$\pi = AcceptanceCriterion(\pi_1, \pi_1, history)$
$\}while(not\ termination)$
$return\ \pi$
$endprocedure$

Figure 11. The general outline for an iterated local search

where $history$ indicates that also the search history may affect the $Perturbation$ and $AcceptanceCriterion$ decisions.

3.2 Variable Iterated Greedy Algorithm

Variable IG algorithm (VIG_FL) is presented and implemented to solve the PFSP with the total tardiness criterion in (Framinan and Leisten, 2008). This algorithm is inspired from the idea of neighbourhood change of the VNS algorithm that is explained in Section 3.5 . In (Ruiz R., Stützle T., 2007) it is shown that destruction of 4 jobs is most adequate, so in their study, the destruction size is used as a constant parameter which equals to 4. In VIG_FL, the destruction size is developed as a variable and at the beginning it is fixed at $d = 1$. If the solution is not improved, the destruction size is incremented by 1 until the maximum destruction size which is $d_{max} = n - 1$ (where n indicates the number of jobs). At any destruction size, if there is an improvement in the solution, destruction size is again fixed at $d = 1$ and search starts all over again. The pseudo code of VIG_FL is given below;

$Procedure\ VIG\_FL$
$\pi = NEH$
$d_{max} = n - 1$
$\pi^R = \pi_b = \pi$
$\pi = RIS(\pi, \pi^R)$
$do\{$
$d = 1$
$do\{$
$\pi_1 = DestructConstruct(\pi, d)$
$\pi_2 = RIS(\pi)$
$if\ \big(f(\pi_2) < f(\pi_1)\big)\ then\{$
$d = 1$
$\pi = \pi^R = \pi_2$

$if\ \left(f(\pi_2) < f(\pi_b)\right)\ then\{$
$\pi_{bsf} = \pi$
$\}endif$
$\}else\ if\ \{$
$U(0,1) \leq \exp\{-(f(\pi_2) - f(\pi))/T\}$
$\pi = \pi^R = \pi_2$
$d = d + 1$
$\}endif$
$\}while(d \leq d_{max})$
$\}while(NotTermination)$
$return\ \pi_b$
$endprocedure$

Figure 12. Variable Iterated Greedy Algortihm of Framinan and Leisten

3.3 Variable Iterated Greedy Algorithm with Differential Evolution

Standard Differential Evolution (DE) algorithm is introduced by (Storn R, Price K., 1997) for continuous optimization problems. DE algorithm is a population-based algorithm and there are three different individual types; target, mutant and trial. At the beginning, population is consisting of "population size" (PS) many target individuals. Mutant individuals are generated by applying *mutation* operation and trial individuals are generated by *crossover* operation and then applies *selection* operator to determine the new target individuals for the next generation.

Let $X_i^t = [x_{i1}^t, x_{i2}^t, .., x_{iD}^t]$ denote the ith individual of target population at generation t. $V_i^t = [v_{i1}^t, v_{i2}^t, .., v_{iD}^t]$ denote mutant and $U_i^t = [u_{i1}^t, u_{i2}^t, .., u_{iD}^t]$ denote trial individuals. Mutant and trial individuals are generated as follows;

$$v_{ij}^t = x_{aj}^{t-1} + F(x_{bj}^{t-1} - x_{cj}^{t-1}) \tag{18}$$

$$u_{ij}^t = \begin{cases} v_{ij}^t & if\ rand < CR\ or\ j = j_r \\ x_{ij}^{t-1} & otherwise \end{cases} \tag{19}$$

where a, b, and c are random integers that are different than each other and i, and take values between $[1, PS]$. $rand$ is a random number between $[0,1)$, F is the mutation scale factor from $[0,2)$ and CR is the crossover probability from $[0,1]$. j_r is a random integer from $[1, D]$. For creating a mutant individual in (18), three different individuals are randomly chosen from target population and the

difference of two of these individuals' jth elements is multiplied with mutation scale factor and added to third individual's jth element. So that, new mutant individual's jth element is obtained. In (19), with crossover probability the trial individual is taken from mutant population, otherwise it remains same as target individual.

By using these formulations, (18) and (19), trial population is generated for $j = 1,2,..,D$ and $i = 1,2,..,PS$. In selection phase, target population and trial population is compared and the one with better objective function value is selected. This phase is performed as;

$$X_i^t = \begin{cases} U_i^t & if\ f(U_i^t) \leq f(X_i^{t-1}) \\ X_i^{t-1} & otherwise \end{cases} \quad (20)$$

where $f(x)$ is the objective function value for individual x. These iterations go on until pre-determined stopping criteria is satisfied.

A modified VIG algorithm is also applied to NIPFS problem to compare the performance of proposed algorithm. (Tasgetiren M.F., Pan Q., Suganthan P.N., Buyukdagli O., 2013) proposed an algorithm that the standard DE algorithm is modified and applied such that the probability to apply IG algorithm to the specific individual in the target population and the parameter of IG, destruction size is a variable. Thus, this algorithm is a variable iterated greedy algorithm guided by differential evolution denoted by VIG_DE.

Basically, VIG_DE algorithm optimizes the probability to apply IG to an individual (ρ_i) and the destruction size (d_i) that is used as a parameter of IG, by using DE. In the initial population, the permutation of the first individual is constructed by using NEH heuristic. All the remaining individuals in the target population are generated randomly and NEH heuristic is applied each of them to start the algorithm with relatively better individuals. The destruction size, d_i, is determined randomly and uniformly between $[1, n-1]$. After generating target population and d_i for each individual of that population, IG algorithm is applied the individuals in target population without considering the probability (ρ_i) that guides algorithm either IG is applied or not. Next, ρ_i is determined as follows;

$$\rho_i = 1 - f(\pi_i) / \sum_{i=1}^{NP} f(\pi_i) \quad (21)$$

A uniform random number $rand$ is generated between [0,1), if this number is less than the probability ρ_i, IG algorithm is applied to the trial individual with the destruction size d_i. This calculation gives a high ratio which means higher probability to apply IG, when the objective function value is lower (for minimization problem) for that individual. This means after applying IG if the objective function value gets better, then the probability to apply IG gets higher value.

In order to avoid complexity, a unique multi-vector chromosome representation is used to keep all variables together in this problem. In Table 2, it can be observed that, the variables d_i and ρ_i appear in the first vector. x_{i1} corresponds to destruction size d_i and x_{i2} to probability ρ_i. The second vector contains the permutation that is assigned to each individual.

j	1	2	3	…	n
x_{ij}	d_i	ρ_i			
π_{ij}	π_{i1}	π_{i2}	π_{i3}	…	π_{in}

Table 3. Multi-vector chromosome representation

In VIG_DE algorithm, mutant individuals are obtained by using formulation (18), for $j = 1,2$ and a, b, and c are randomly chosen integers by tournament selection with size of 2 that are different than each other and i, and take values between $[1, PS]$.

For the crossover phase, an arithmetic crossover operator is applied to generate trial population.

$$u_{ij}^t = C_r(v_{ij}^t) + (1 - C_r)x_{ij}^t \tag{22}$$

where C_r is a crossover probability from the range $[0,1]$ and $j = 1,2$. The higher C_r value means, the higher effect of mutant individual comparing to target individual on the new trial individual. This arithmetic calculation may cause the individual to violate the search range. In order to fix this problem, following formulation is used;

$$u_{ij}^t = x_{ij}^{max} + r(x_{ij}^{max} - x_{ij}^{min}) \tag{23}$$

where $j = 1,2$ and $x_{i1}^{min} = 1, x_{i1}^{max} = n - 1, x_{i2}^{min} = 0$, $x_{i2}^{max} = 1$ and r is a uniform random number from [0,1]. Since the first dimension is taken as a destruction size, this value should be an integer value. Therefore, destruction size is obtained by truncating u_{i1}^t such that; $d_i = \lceil u_{i1}^t \rceil$. The second dimension is used as the probability to apply IG algorithm, $u_{i1}^t = \rho_i$. If a uniform random number r is less than the probability $u_{i1}^t = \rho_i$, then the IG algorithm is applied and the fitness value of the generated trial individual is computed. In the selection phase, the survival of the fittest among all the trial and target individuals is considered as shown below;

$$x_i^t = \begin{cases} u_i^t & if\ f(u_i^t) < f(x_i^{t-1}) \\ x_i^{t-1} & otherwise \end{cases} \quad (24)$$

Differential evolution part of the algorithm, that is explained above, is applied only the first vector of the solution representation which contains x_{i1} and x_{i2}. The pseudo code of the whole VIG_DE algorithm is given in Figure XXX.

$Procedure\ VIG\_DE$
$Initilize\ population$
$Evaluate\ population\ and\ determine\ d_i\ and\ \rho_i$
$While\ (NotTermination)\ do\{$
  $For\ (i = 1\ to\ PS)\ do\{$
    $Get\ mutant\ individual\ v_i$
    $Get\ trial\ individual\ u_i$
    $if(r < \rho_i = u_{i2})then\ \{$
      $Apply\ DestructConstruct\ with\ d_i = u_{i1}\ to\ \pi_i$
      $Apply\ RIS\ local\ search\ to\ \pi_i$
      $Update\ x_i, \pi_i\ and\ \pi_b$
    $\}endif$
  $\}\ endfor$
$\}endwhile$
$return\ \pi_b$
$endprocedure$

Figure 13. Variable Iterated Greedy Algorithm with Differential Evolution

3.4 Hybrid Discrete Differential Evolution

The DE algorithm is proposed for continuous optimization problems where the individuals are represented by floating-point numbers, so in order to apply this algorithm to the problems where discrete job permutation is needed to be generated. (Tasgetiren, M. F., Pan, Q. -K., Liang, Y. -C., Suganthan, P.N., 2007a) proposed an algorithm for scheduling problems which is called Discrete

Differential Evolution (DDE) algorithm. Mutation and crossover operations are re-designed as job-permutation-based that is applicable for discrete cases.

The Hybrid DDE (HDDE) algorithm is the combination of DDE-based evolutionary searching technique and a problem specific local search. (Deng G and Gu X., 2012) inspired from (Tasgetiren M.F., Pan Q.K., Suganthan P.N., Liang Y.C., 2007b) study and applied their perturbed local search after the new population generated by using DDE which provides algorithm to start the new search with a qualified individuals. HDDE algorithm that (Deng G and Gu X., 2012) proposed is one of the metaheuristic algorithms that is compared with the algorithm presented in this study.

In this algorithm, the individuals are represented as job permutations; $\pi = \big(\pi(1), \pi(2), .., \pi(n)\big)$. Different than standard DE that is explained in Section 3.3 , ith individual of target population at generation t is represented as π_i^t. Mutant and trial individuals are generated as follows;

$$V_i^t = \begin{cases} insert\big(\pi_g^{t-1}\big) & if\ rand < p_m \\ insert(\pi_r^{t-1}) & otherwise \end{cases} \tag{25}$$

$$U_i^t = \begin{cases} CR(V_i^t, \pi_i^{t-1}) & if\ rand < p_c \\ V_i^t & otherwise \end{cases} \tag{26}$$

where π_g^{t-1} is a relatively better solution than current one, from generation $t-1$. $insert(x)$ operator is a random insert move in an individual x, $CR(x, y)$ represents a crossover operator (partially mapped crossover) applied to x and y, p_m is the insert mutation scale factor and p_c is the crossover probability. r is a random number between $[1, PS]$ but different than i and $rand$ is a uniform random number between [0,1). In (25), mutant individual is obtained as following; if $rand$ is less than p_m, insertion move is applied to a relatively better solution π_g, otherwise to a random target individual different than ith. Similarly, for the trial individual generation in (26), if $rand$ is less than p_c, partially mapped crossover is applied to mutant individual V_i^t and the corresponding target value at the previous generation π_i^{t-1}, otherwise mutant individual is directly taken.

Selection phase of HDDE is performed same as in standard DE that is represented in (20) in Section 3.3 .

As a local search, in (Deng G and Gu X., 2012) perturbed local search is presented. This local search algorithm is a combination of destruction construction algorithm that is explained in Section 3.1.2 , referenced insertion algoritm that is presented in Section 3.1.3 and *IterativeImprovement_Insertion* that is represented in (Ruiz R., Stützle T., 2007). The outline of perturbed local search is shown below;

$Procedure\ PLS(\pi)$
$\pi_1 = \pi$
$\pi_2 = DestructionConstruction(\pi_1)$
$\pi_3 = InsertionImprovement(\pi_2)$
$if\big(f(\pi_3) < f(\pi)\big)then\ \{$
$\pi = \pi_3$
$if\big(f(\pi) < f(\pi_b)\big)then\ \{$
$\pi_b = \pi$
$\}endif$
$\}\ else\ if\ rand < \exp\{-\big(f(\pi_3) - f(\pi)\big)/T\}\ then\ \{$
$\pi = \pi_3$
$\}endif$
$endprocedure$
$return\ \pi$

Figure 14. Perturbed Local Search

where π_b is the best individual found by HDDE so far. For a given permutation π, first destruction construction is applied, then insertion-based local search and lastly the acceptance criterion is checked.

The initial target population is generated randomly except two individuals of this population. One is generated by using NEH heuristic that is explained in Section 3.1.1, the other is generated by a variant of *InsertionImprovement(π).*

The pseudo code of HDDE algorithm is given below. This algorithm includes standard DE procedures, mutation, crossover and selection and as a local search perturbed local search.

$Procedure\ HDDE$
$set\ parameters: p_c, p_m, PS, d, T, t = 1$
$initialize\ target\ population$
$\pi_{bsf} = the\ best\ individual\ in\ target\ population, \pi_g = \pi_b$
$while(not\ termination)$
$t = t + 1$
$obtain\ mutant\ population$
$obtain\ trial\ population$
$selection$

$update\ \pi_b$
$PLS(\pi_g)$
$endwhile$
$endprocedure$

Figure 15. Hybrid Discrete Differential Evolution

3.5 General Variable Neighbourhood Search

Variable neighborhood search (VNS) is a common approach to enhance the solution quality with systematic changes of neighborhood within a local search. It is proposed by (Mladenovic´, N., Hansen, P., 1997). The algorithm involves iterative exploration of larger and larger neighborhoods for a given local optima until there is an improvement, after which time the search is repeated. The basic steps of VNS algorithm can be summarized as given below:

Initially, a set of neighborhood structures, N_k is selected where $k = 1,2,..,k_{max}$. Having a multi neighborhood structure makes VNS an effective algorithm since most local search heuristics use one structure, $k_{max} = 1$. Then the initial solution is generated either randomly or using heuristics, such as NEH heuristic. The stopping criteria can be selected as maximum CPU time allowed or maximum number of iterations. Then, following steps are repeated until the stopping criterion is met;

- Set $k = 1$;
- Repeat the following steps until $k > k_{max}$:
  - Generate a point x' at random from k^{th} neighborhood of x , $(x' \in N_k(x))$, $(shaking)$
  - Apply local search method by considering x' as initial solution and obtain a local optimum denoted by x''. (*local search*)
  - If this local optimum x'' is better than x, $x = x''$ and continue search with current neighborhood structure N_1 meaning $k = 1$; otherwise set $k = k + 1$.

There are some decisions to be made before using VNS algorithm. These are;

- Number and types of neighborhoods to be used
- Order of their use in the search
- Strategy for changing the neighborhoods
- Local search method
- Stopping condition

Shaking step of VNS algorithm provides randomness in search. If this step is eliminated from algorithm, variable neighborhood descent (VND) algorithm is obtained. The steps of VND can be explained briefly, as follows;

- Set $k = 1$;
- Repeat the following steps until $k > k_{max}$:
  - Find the best $x' \in N_k(x)$
  - If x' is better than x, then set $x = x'$; otherwise set $k = k + 1$.

An extended VNS algorithm called general variable neighborhood search (GVNS) that is proposed in (Hansen P., Mladenovic N., Urosevic D., 2006). It can be obtained by replacing the *local search* step of VNS with VND algorithm. In this study, a different version of the GVNS algorithm with insert and swap operations in outer loop and in the inner loop (VND), IG algorithm and iterated local search (ILS) algorithm is applied to NIPFS problem and compared to all other algorithms. Using an another algorithm in local search step of the VNS provides to have a more powerful algorithm.

The pseudo code of the GVNS algorithm we applied in this study is given below;

$Procedure\ GVNS$
$\pi = NEH$
$\pi_b = \pi$
$k_{max} = 2$
$k = 1$
$do\{$
    $\pi_1 = N_k(\pi)$
    $\pi_2 = VND(\pi_1)$
    $If\ f(\pi_2) < f(\pi)$
        $\pi = \pi_2$
        $k = 1$
    $else$
        $k = k + 1$

    $\}while(k \leq k_{max})$
$return\ \pi_b$
$endprocedure$

Figure 16. General Variable Neighborhood Search Algorithm

where, $N_1(\pi) = insert(\pi)$ and $N_2(\pi) = swap(\pi)$ operations. Shaking phase is composed of two different operations. The $insert$ and $swap$ operations applies only one insert and one swap move, respectively, to *shake* the permutation. The sequence of these operations has an

important role in search. In literature many study shows that putting insert operation before swap results better than the converse version. After shaking phase $VND(\pi)$ is applied, as a local search, and is explained below;

$Procedure\ VND(\pi)$
$d_{max} = 2$
$d = 1$
$do\{$
$\pi_1 = N'_d(\pi)$
$If\ f(\pi_1) < f(\pi)$
$\pi = \pi_1$
$d = 1$
$else$
$d = d + 1$
$\}while(d \leq d_{max})$
$return\ \pi$
$endprocedure$

Figure 17. Variable Neighborhood Descent

where $N_1'(\pi) = IG(\pi)$ which is explained in Section 3.1 but with some differences. These differences will be shown in pseudo code given below. $N_2'(\pi) = ILS(\pi)$, that will be explained briefly in Section 3.1.4 .

In $VND(\pi)$ phase, IG algorithm is applied until there is no improvement. After the neighborhood structure is changed as ILS algorithm. ILS algorithm is also applied as long as there is improvement. Otherwise the search is stopped.

$Procedure\ N'_1(\pi)$
$\pi_1 = DestructConstruct(\pi)$
$Flag = true$
$do\{$
$\pi_2 = RIS(\pi_1)$
$If\ f(\pi_2) < f(\pi_1)$
$\pi_1 = \pi_2$
$Flag = true$
$else$
$Flag = false$
$\}while(Flag = true)$
$return\ \pi_1$
$endprocedure$

Figure 18. The first neighborhood structure of VND in GVNS

In this study, as mentioned before, ILS algorithm is used as a second neighborhood structure in VND phase. Using that much powerful algorithm as a neighborhood structure instead of more basic ones, increase this phase's ability to

reach better solutions. According to the No-Free Lunch (NFL) theorem that is proposed in (Wolpert D. H. and Macready W. G., 1997); "For any algorithm, any elevated performance over one class of problems is offset by performance over another class". Meaning that, any algorithms performance may differ from problem to problem. So applying two different algorithms to a specific problem and changing the neighborhood while there is no improvement may increase the chance to get better solution. At some point, one of the algorithms may stuck and perform worse, when the neighborhood is changed, the other algorithm may perform very well.

$Procedure\ N'_2(\pi)$
$\pi_1 = perturbation(\pi)$
$Flag = true$
$do\{$
    $\pi_2 = RIS(\pi_1)$
    $If\ f(\pi_2) < f(\pi_1)$
        $\pi_1 = \pi_2$
        $Flag = true$
    $else$
        $Flag = false$
    $\}while(Flag = true)$
$return\ \pi_1$
$endprocedure$

Figure 19. The second neighborhood structure of VND in GVNS

where $perturbation$ has a variable input that is called *perturbation size* and selected randomly between [1,5] as applied and shown in (Pan, Q-K. and Wang, L., 2008) study that these interval results better than other. The perturbation is applied to permutation π, that is obtained from the first neighborhood structure of VND. *Perturbation strength* many inserts are made in this step. Then, the local search RIS, that is explained in Section 3.1.3, is applied to newly generated permutation π_1 until there is no improvement.

CHAPTER 4

COMPUTATIONAL RESULTS

In this thesis, different algorithms that are proposed to solve no-idle permutation flow shop scheduling problem is compared with a newly modified GVNS algorithm, as mentioned in previous chapters. In Chapter 3, some metaheuristics that are applied to NIPFS problem from the literature are explained. Then, the computational results of these algorithms are compared with the GVNS algorithm.

In order to test the performance of these algorithms, the benchmark suite presented in the personal website of Ruiz García, Rubén[1] is used. This benchmark is designed for NIPFS problem with makespan criterion specifically, with the number of jobs $n = \{50,100,150,200,250,300,350,400,450,500\}$ and the number of machines $m = \{10,20,30,40,50\}$. There are 50 combinations with different sizes and each combination has 5 different instances. Thus, there are 250 instances in total. 5 runs were carried out for each instance for each algorithm. All results are compared with the best-known solutions presented in the website of Ruiz García, Rubén[1]. In order to compare these results, an average relative percentage deviation is calculated for each combination by using the following equation;

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{100(H_i - Best)}{Best} \right) \Big/ R \qquad (26)$$

where H_i is the objective function value that is obtained in i^{th} run of each algorithm, $Best$ is the best-known solution presented in the website of Ruiz García, Rubén[1] and R is the number of runs. The stopping criterion is selected as a maximum run time of each algorithm which is defined as $T_{max} = n(m/2) \times t$ milliseconds where the value of t can be taken as, $t = 30$ or $t = 60$ depending on the comparison case. In the comparison tables, $AlgortihmName^t$ denotes that the algorithm "$AlgortihmName$" was run for $T_{max} = n(m/2) \times t$.

The proposed algorithms are coded in C++ and run on an Intel Core 2 Quad 2.66 GHz PC with 3.5 GB memory. The parameters of DE, the crossover

[1] http://soa.iti.es/rruiz

probability and mutation scale factor are taken as $CR = 0.9$ and $F = 0.9$, respectively. These high ratios provide algorithms to enhance the search space and to increase the diversification of solutions. The population size is taken as $PS = 30$. For the Destruction and Construction procedure, the destruction size is fixed at $d = 4$.

The results of HDDE algorithm are taken directly from (Deng G and Gu X., 2012) study. They implemented the proposed algorithm HDDE to NIPFS problem with termination criterion of $T_{max} = n(m/2) \times t$ milliseconds where $t = 60$. There are a few differences in properties of their computer and the computer that is used in this study. These differences may cause an unfair comparison, therefore, in order to avoid this inequality, the algorithms other than HDDE were run with termination criterion $T_{max} = n(m/2) \times t$ milliseconds where $t = 30$. In other words, all other algorithms were run for half of the CPU time that HDDE algorithm was run for. The computational results are given in Table 4.

From Table 4, it can be observed that the proposed algorithm, GVNS, has better average relative percentage deviations than the other algorithms that applied to NIPFS problem. The GVNS algorithm was able to further improve the *Best* results to -0.213 which indicates that this algorithm is superior than the other four algorithms.

Table 4. Average relative percentage deviation of the algorithms

Jobs	Machines	HDDE[60]	vIG_DE[30]	IG_RIS[30]	VIG_FL[30]	GVNS[30]
50	10	0.20	0.03	0.04	0.08	0.14
	20	0.29	-0.04	-0.06	0.04	-0.04
	30	0.25	-0.17	0.08	0.06	-0.12
	40	0.36	-0.41	0.13	-0.10	-0.41
	50	1.15	-0.16	1.06	0.56	-0.23
100	10	0.10	0.04	0.06	0.06	0.08
	20	0.09	-0.09	0.07	-0.04	-0.03
	30	0.50	-0.19	-0.17	0.05	-0.30
	40	0.07	-0.65	-0.41	-0.41	-0.95
	50	0.45	-0.12	0.35	0.49	-0.27
150	10	0.01	0.00	0.00	0.01	0.00
	20	0.43	0.05	0.04	0.12	0.02
	30	0.14	-0.18	0.01	-0.07	-0.12
	40	0.25	-0.07	-0.05	0.29	-0.25

Jobs	Machines	HDDE[60]	vIG_DE[30]	IG_RIS[30]	VIG_FL[30]	GVNS[30]
	50	0.17	-0.85	-0.46	-0.57	-0.97
200	10	0.03	0.00	0.00	0.00	-0.00
	20	0.04	-0.07	0.00	-0.01	-0.05
	30	0.01	-0.31	-0.17	-0.20	-0.33
	40	0.10	-0.26	-0.29	-0.26	-0.61
	50	0.45	-0.40	-0.22	-0.17	-0.53
250	10	0.00	-0.01	0.00	0.00	-0.01
	20	0.13	-0.03	0.05	0.02	-0.01
	30	0.00	-0.14	-0.06	-0.12	-0.28
	40	0.31	0.02	0.04	0.11	-0.14
	50	0.06	-0.60	-0.77	-0.49	-1.09
300	10	0.00	0.00	0.00	0.00	0.00
	20	0.12	0.00	-0.03	-0.03	-0.05
	30	0.30	0.02	0.00	-0.06	-0.05
	40	0.15	-0.34	-0.04	-0.17	-0.32
	50	0.10	-0.23	-0.27	-0.15	-0.54
350	10	0.02	0.00	0.02	0.00	0.01
	20	0.05	-0.01	0.02	-0.01	0.00
	30	0.11	-0.05	-0.01	-0.08	-0.14
	40	0.31	0.08	0.02	0.01	-0.17
	50	0.18	-0.44	-0.40	-0.38	-0.72
400	10	0.01	0.00	0.00	0.00	-0.00
	20	0.14	0.04	0.06	-0.01	0.00
	30	0.23	0.11	0.05	-0.01	-0.02
	40	0.20	-0.04	-0.09	-0.15	-0.16
	50	0.08	-0.25	-0.28	-0.33	-0.49
450	10	0.02	0.00	0.00	0.00	0.01
	20	0.12	0.00	0.04	0.00	0.04
	30	0.08	-0.03	-0.09	-0.12	-0.16
	40	0.01	-0.09	-0.02	-0.23	-0.23
	50	0.21	-0.07	-0.27	-0.43	-0.53
500	10	0.01	0.00	0.01	0.00	0.00
	20	0.04	-0.04	0.00	-0.04	-0.04
	30	0.13	0.08	0.04	0.00	-0.05
	40	0.13	0.10	-0.04	-0.04	-0.17
	50	0.17	-0.03	-0.08	-0.23	-0.38
Overall Avg		0.16	**-0.12**	**-0.04**	**-0.06**	**-0.213**

In order to determine if the average relative percentage deviations of algorithms are statistically significant, an interval plot is given in Figure 20. Vertical lines with horizontal lines at their end points represent the 95% confidence interval for the mean and the symbol at the middle indicates the mean of each algorithm's relative percentage deviations. As can be observed from the interval plot, 95% confidence interval for the mean of GVNS algorithm is obviously does not coincide with the others which means that the means are statistically significant. In addition the mean of GVNS is significantly lower than the others.

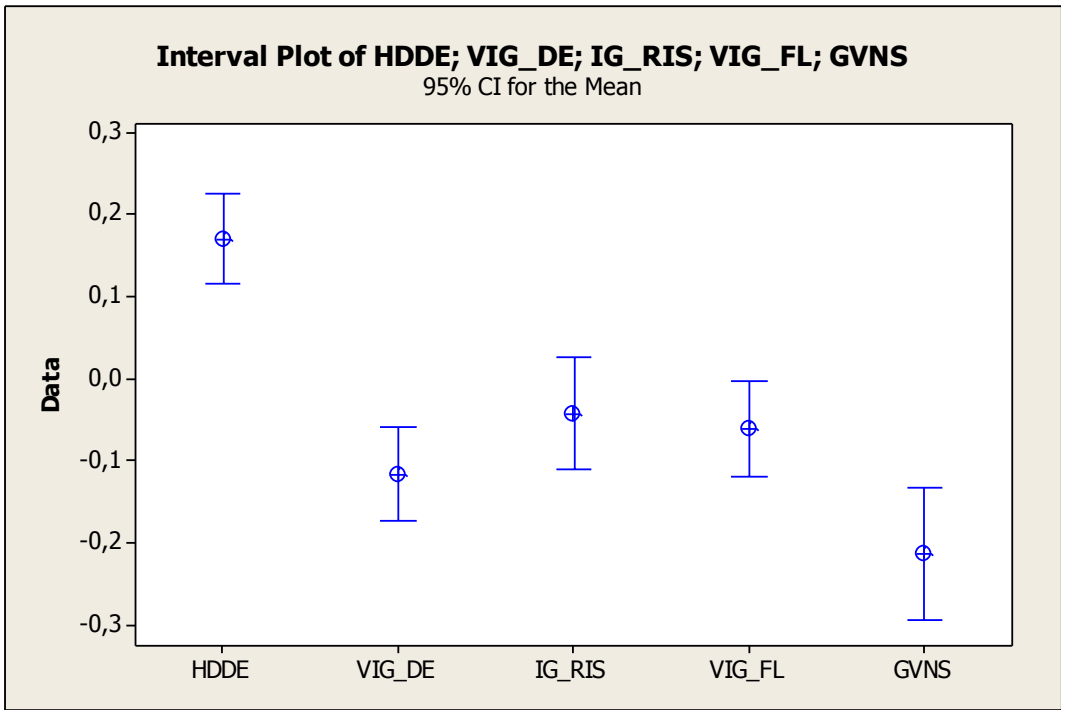


Figure 20. Interval plot of algoritms compared

In Table 5, the best makespan values obtained by applying the compared algorithms to the benchmark suits are presented. The first three columns represent the number of jobs, number of machines and the instance number of the problem solved in that row, respectively. The column "Best" represents the best-known solutions presented in the website of Ruiz García, Rubén. The values in column HDDE[60] are directly taken from (Deng G and Gu X., 2012), the remaining results obtained by applying these algorithms to the problem. For each row, meaning an instance, the minimum result is represented in bold font.

Table 5. Makespan values obtained by the algorithms

Jobs	Machines	Instances	Best	HDDE^{60}	VIG_DE^{60}	IG_RIS^{60}	VIG_FL^{60}	GVNS^{60}
50	10	1	**4127**	**4127**	**4127**	**4127**	**4127**	**4127**
		2	**4283**	**4283**	**4283**	**4283**	**4283**	**4283**
		3	**3262**	3263	**3262**	**3262**	**3262**	3267
		4	3219	**3216**	**3216**	3219	3219	3219
		5	**3470**	**3470**	**3470**	3471	3471	**3470**
	20	1	5647	5647	5647	5647	5647	**5646**
		2	5834	5820	5818	5820	5820	**5814**
		3	5794	**5793**	**5793**	**5793**	**5793**	**5793**
		4	5803	5798	5799	5798	**5795**	5798
		5	4907	4881	4884	4900	4897	**4874**
	30	1	7243	7256	**7223**	7239	7239	7242
		2	7381	7351	7351	7331	**7330**	7334
		3	6902	**6844**	6857	6900	6885	6850
		4	7624	**7579**	**7579**	7580	7580	7585
		5	7340	7338	**7333**	7366	7366	7337
	40	1	9264	9227	9168	9167	**9130**	9171
		2	10164	**10116**	10137	10121	10117	10134
		3	9896	9791	**9782**	9854	9836	9822
		4	9575	9607	9523	9550	9533	**9495**
		5	9082	8967	8968	8960	8957	**8904**
	50	1	11652	11717	11604	11753	11753	**11584**
		2	10946	10980	10893	10942	10942	**10857**
		3	10960	10960	10885	10955	10935	**10873**
		4	10026	10044	9967	10030	10030	**9890**
		5	11380	11349	**11316**	11365	11348	11359
100	10	1	6575	**6570**	**6570**	6575	6575	6575
		2	**5798**	5802	5803	5808	5802	5802
		3	**6533**	**6533**	**6533**	**6533**	**6533**	**6533**
		4	6161	6171	**6158**	**6158**	**6158**	**6158**
		5	**6654**	**6654**	**6654**	**6654**	**6654**	**6654**
	20	1	8611	**8606**	**8606**	**8606**	**8606**	**8606**
		2	8223	**8218**	8224	8241	8241	8235
		3	9057	9057	**9043**	9055	**9043**	**9043**
		4	9031	9029	8972	8973	**8970**	**8970**
		5	9126	9125	**9109**	**9109**	**9109**	**9109**
	30	1	11249	11228	11210	11210	**11202**	**11202**
		2	10989	10943	**10938**	**10938**	**10938**	10943
		3	10666	10674	10571	10555	10555	**10549**

Jobs	Machines	Instances	Best	HDDE^{60}	VIG_DE^{60}	IG_RIS^{60}	VIG_FL^{60}	GVNS^{60}
		4	11175	11137	11103	**11097**	**11097**	**11097**
		5	11030	11065	**10983**	10996	10985	10986
	40	1	12806	12721	**12606**	12640	12640	12608
		2	13306	13295	**13117**	13206	13202	13162
		3	12654	12574	12488	12528	12504	**12411**
		4	12044	11934	**11781**	11857	11829	11778
		5	12934	12911	12920	12919	12919	**12902**
	50	1	16111	16035	16019	16057	16050	**15998**
		2	15019	14800	**14787**	14989	14916	14853
		3	17755	17798	17585	17686	17650	**17571**
		4	16672	16703	16684	16692	16667	**16626**
		5	14827	14948	14802	14920	14854	**14746**
150	10	1	**10404**	**10404**	**10404**	**10404**	**10404**	**10404**
		2	**8824**	**8824**	**8824**	8826	**8824**	8826
		3	**9180**	**9180**	**9180**	**9180**	9181	**9180**
		4	**10032**	**10032**	**10032**	**10032**	**10032**	**10032**
		5	9870	9870	**9866**	9870	9870	9870
	20	1	10768	10823	**10758**	10800	10790	10771
		2	11718	11725	**11699**	11704	**11699**	**11699**
		3	12063	12058	**12046**	12063	12060	**12046**
		4	10965	11001	10936	10933	10933	**10903**
		5	**13210**	**13210**	**13210**	**13210**	**13210**	**13210**
	30	1	15569	15548	15505	15500	15500	**15482**
		2	13747	13719	**13667**	13699	**13667**	**13667**
		3	14688	14688	14651	14673	14651	**14650**
		4	14627	14574	**14549**	**14549**	**14549**	**14549**
		5	**15257**	15259	15265	15277	15277	15276
	40	1	16217	16114	16025	16135	16120	**15935**
		2	18235	18238	18122	18218	18173	**18075**
		3	16416	16434	**16356**	16419	16381	16375
		4	14658	14647	14648	14651	14640	**14555**
		5	17298	17260	17246	17288	17244	**17234**
	50	1	20625	20388	20364	20371	20367	**20298**
		2	19512	19389	**19121**	19247	19227	19173
		3	19702	19655	19447	19476	**19358**	19419
		4	20355	20166	**20139**	20237	20224	20143
		5	19611	19342	19308	19429	19418	**19241**
200	10	1	**12155**	**12155**	**12155**	**12155**	**12155**	**12155**

Jobs	Machines	Instances	Best	HDDE[60]	VIG_DE[60]	IG_RIS[60]	VIG_FL[60]	GVNS[60]
		2	**12227**	**12227**	**12227**	**12227**	**12227**	**12227**
		3	**12595**	**12595**	**12595**	**12595**	**12595**	**12595**
		4	12304	**12301**	**12301**	**12301**	**12301**	**12301**
		5	**12076**	**12076**	**12076**	**12076**	**12076**	**12076**
	20	1	**14864**	14877	**14864**	**14864**	**14864**	**14864**
		2	14134	14093	14095	14126	**14086**	**14086**
		3	16135	**16115**	**16115**	**16115**	**16115**	**16115**
		4	**15972**	**15972**	**15972**	**15972**	**15972**	**15972**
		5	14225	14214	**14175**	14211	14211	14190
	30	1	17222	17116	17053	17044	17044	**17034**
		2	17126	**16972**	16994	17019	17014	16975
		3	17529	17501	17428	17428	**17420**	**17420**
		4	20032	20020	19991	**19986**	**19986**	20008
		5	17995	18077	17974	17988	17982	**17970**
	40	1	20128	20024	19965	19953	19953	**19909**
		2	21801	21743	21724	21797	21763	**21708**
		3	20609	20642	**20564**	20629	20624	20607
		4	17864	17624	17507	17628	17548	**17420**
		5	21258	21234	21237	21216	**21209**	21217
	50	1	22912	22959	22729	22632	**22580**	22656
		2	23664	23631	23488	23528	23509	**23429**
		3	22615	22561	22431	22471	22438	**22296**
		4	24140	24221	23969	24092	24022	**23929**
		5	24424	24334	**24275**	24374	24355	24321
250	10	1	16640	16640	**16639**	16640	16640	**16639**
		2	15483	**15476**	**15476**	**15476**	**15476**	**15476**
		3	**14872**	**14872**	**14872**	**14872**	**14872**	**14872**
		4	**15247**	15248	**15247**	15250	15250	**15247**
		5	**15026**	**15026**	**15026**	**15026**	**15026**	**15026**
	20	1	17613	17633	**17577**	17583	17578	17593
		2	17692	17684	**17683**	**17683**	**17683**	**17683**
		3	17534	17543	17500	17522	**17487**	17496
		4	17651	17646	17645	17647	17647	**17639**
		5	**17274**	17297	17277	17277	17277	17282
	30	1	**21920**	**21920**	**21920**	**21920**	**21920**	**21920**
		2	21946	21876	21853	21890	21884	**21824**
		3	20196	20096	20111	20107	**20077**	20082
		4	19886	19807	19794	19821	19809	**19744**

Jobs	Machines	Instances	Best	HDDE^{60}	VIG_DE^{60}	IG_RIS^{60}	VIG_FL^{60}	GVNS^{60}
		5	20991	20910	**20906**	20909	20907	20941
	40	1	22871	22870	22820	22957	22899	**22780**
		2	24193	24247	24119	24152	24149	**24098**
		3	24412	24482	24353	24394	24394	**24337**
		4	24913	24876	24748	24824	24824	**24741**
		5	23536	23562	23506	23478	23471	**23468**
	50	1	28662	28898	28563	28610	28599	**28548**
		2	24932	24930	24577	24770	24480	**24276**
		3	26973	26678	26512	26563	26528	**26351**
		4	25971	25565	25611	25612	25608	**25500**
		5	27670	27542	27389	27393	27393	**27332**
300	10	1	**17498**	**17498**	**17498**	**17498**	**17498**	**17498**
		2	**17350**	**17350**	**17350**	17351	17351	**17350**
		3	**18627**	**18627**	**18627**	18628	**18627**	**18627**
		4	**16941**	**16941**	**16941**	**16941**	**16941**	**16941**
		5	17524	17524	**17521**	17524	17524	**17521**
	20	1	18873	18884	**18837**	18870	18859	18840
		2	**22032**	22035	**22032**	**22032**	**22032**	**22032**
		3	20268	20278	20230	20235	20235	**20229**
		4	19516	19497	19490	**19484**	**19484**	19490
		5	**20705**	20707	**20705**	**20705**	**20705**	**20705**
	30	1	26565	26529	26501	26530	26530	**26487**
		2	24290	24350	24267	**24261**	**24261**	24269
		3	24386	24381	24370	24369	24369	**24363**
		4	23728	23761	**23710**	23727	23726	23726
		5	22638	22630	22568	22558	22558	**22553**
	40	1	26817	26762	**26599**	26640	26639	26600
		2	29332	29316	**29158**	29175	29173	29180
		3	25534	25391	25362	25598	25565	**25361**
		4	27648	27565	27479	27498	27498	**27466**
		5	28872	28812	**28760**	28810	28810	28822
	50	1	31860	31755	31667	31775	31683	**31538**
		2	29834	29515	29490	29676	29657	**29474**
		3	30867	30892	30731	30747	30747	**30701**
		4	32402	32405	**32265**	32350	32346	32286
		5	29512	29359	**29051**	29239	29239	29110
350	10	1	19302	19300	19302	19302	19302	**19297**
		2	21319	21319	**21316**	21319	21320	21318

Jobs	Machines	Instances	Best	HDDE[60]	VIG_DE[60]	IG_RIS[60]	VIG_FL[60]	GVNS[60]
		3	**21330**	**21330**	**21330**	**21330**	**21330**	**21330**
		4	**21759**	**21759**	**21759**	**21759**	**21759**	**21759**
		5	**20591**	**20591**	**20591**	**20591**	**20591**	**20591**
	20	1	25417	25417	**25413**	25415	**25413**	25417
		2	**27185**	**27185**	**27185**	**27185**	**27185**	**27185**
		3	22906	22906	**22880**	22907	22907	**22880**
		4	22994	22985	**22968**	22971	22970	**22968**
		5	22778	22750	**22746**	22750	22750	22756
	30	1	25382	25393	**25226**	25355	25275	25275
		2	27773	27812	27744	**27740**	**27740**	27741
		3	27775	27673	27653	27657	27657	**27638**
		4	29358	29306	**29295**	29295	29295	29297
		5	25240	**25209**	25230	25221	25211	25211
	40	1	29381	29282	29182	29241	29241	**29072**
		2	29163	29239	29043	29024	29024	**29010**
		3	36287	36368	**36247**	36253	36249	36252
		4	34788	34744	**34644**	34677	34669	34692
		5	29847	29905	29840	29879	29814	**29742**
	50	1	32559	32375	**32144**	32435	32178	32167
		2	33454	33167	32911	33042	33032	**32882**
		3	34982	34908	34718	34735	34697	**34682**
		4	37210	37081	37009	37031	36996	**36985**
		5	35710	35588	35390	35349	**35348**	35363
400	10	1	25244	25238	**25238**	**25238**	**25238**	**25238**
		2	**23001**	**23001**	**23001**	**23001**	**23001**	**23001**
		3	**23665**	**23665**	**23665**	**23665**	**23665**	**23665**
		4	**23275**	23277	**23275**	**23275**	**23275**	**23275**
		5	**21956**	**21956**	**21956**	**21956**	**21956**	**21956**
	20	1	27704	27696	**27686**	**27686**	**27686**	**27686**
		2	28092	28092	**28088**	28092	28092	28089
		3	26254	26274	**26224**	**26224**	**26224**	26227
		4	25164	25177	25155	25168	25168	**25105**
		5	24753	24711	**24688**	24707	24702	24690
	30	1	29487	29473	**29405**	29446	29444	29467
		2	29295	29296	**29217**	29274	29241	29275
		3	28725	28793	28733	28691	**28633**	28657
		4	31324	31381	31279	31281	**31278**	31282
		5	34543	34579	34535	34534	**34533**	34539

Jobs	Machines	Instances	Best	HDDE^{60}	VIG_DE^{60}	IG_RIS^{60}	VIG_FL^{60}	GVNS^{60}
	40	1	37552	37587	37440	37498	37464	**37426**
		2	33936	33912	**33805**	33839	33839	33811
		3	34548	34498	34482	34499	34491	**34450**
		4	35406	35403	35306	35262	**35245**	35247
		5	32853	32936	32739	32777	32776	**32730**
	50	1	38000	38006	**37825**	37871	37827	37849
		2	38411	38342	**38237**	38275	38275	38283
		3	38337	38122	37880	37930	37912	**37680**
		4	40578	40521	40465	40523	40480	**40444**
		5	35968	35921	35516	35628	35527	**35499**
450	10	1	23989	23989	**23987**	**23987**	**23987**	**23987**
		2	**26277**	**26277**	**26277**	**26277**	**26277**	**26277**
		3	**25849**	**25849**	**25849**	**25849**	**25849**	**25849**
		4	**26910**	26911	**26910**	26911	**26910**	**26910**
		5	**25191**	**25191**	**25191**	**25191**	**25191**	**25191**
	20	1	27533	27522	**27514**	27527	27521	27519
		2	**27924**	27930	**27924**	27930	**27924**	**27924**
		3	28811	28808	28770	28772	**28769**	28774
		4	28485	28494	**28446**	28459	**28446**	28457
		5	28543	28585	28541	28541	**28539**	28540
	30	1	35197	35222	35127	**35123**	**35123**	35152
		2	32639	32544	32517	32565	32561	**32494**
		3	32131	32057	32021	32036	32001	**31968**
		4	33769	33754	33700	33731	33731	**33691**
		5	33732	33755	33667	33664	33658	**33641**
	40	1	39797	39677	39562	39596	39557	**39547**
		2	36174	36137	**36020**	36050	36050	36045
		3	37917	**37794**	37811	37829	37829	37817
		4	37774	37770	37606	37610	**37596**	37674
		5	35870	35773	35712	35731	35731	**35681**
	50	1	37913	37847	37563	37442	37442	**37287**
		2	43449	43503	**43392**	43447	43431	43461
		3	44275	44157	44087	**44010**	**44010**	44152
		4	41295	41421	**41036**	41044	41044	41040
		5	41348	41180	**40923**	41047	41044	40931
500	10	1	28839	28840	**28839**	28840	28840	**28839**
		2	27924	**27923**	**27923**	**27923**	**27923**	27925
		3	**27349**	**27349**	**27349**	**27349**	**27349**	**27349**

Jobs	Machines	Instances	Best	HDDE^{60}	VIG_DE^{60}	IG_RIS^{60}	VIG_FL^{60}	GVNS^{60}
		4	**27575**	**27575**	**27575**	**27575**	**27575**	**27575**
		5	**27457**	**27457**	**27457**	**27457**	**27457**	**27457**
	20	1	35973	**35948**	**35948**	**35948**	**35948**	**35948**
		2	34134	**34129**	**34129**	**34129**	**34129**	**34129**
		3	31114	31102	31066	31069	**31065**	31071
		4	30916	30905	**30900**	**30900**	**30900**	**30900**
		5	33776	33782	**33768**	**33768**	**33768**	33776
	30	1	36381	36417	**36337**	36345	36345	36358
		2	39381	39367	39357	39357	39356	**39348**
		3	39261	39290	**39226**	39256	39250	39273
		4	34003	33972	33918	33942	33927	**33896**
		5	38368	38390	38340	38353	38348	**38339**
	40	1	40793	40768	40708	40732	**40685**	40747
		2	44170	44181	**44099**	44144	44144	44148
		3	40523	40485	40366	40357	40357	**40313**
		4	41992	42094	41917	**41886**	**41886**	41907
		5	36448	36343	36312	36449	36351	**36203**
	50	1	46461	46331	46238	46268	46263	**46175**
		2	43461	43552	**43281**	43379	43379	43345
		3	45484	45409	45206	**45164**	**45164**	45193
		4	42620	42687	42417	42454	42454	**42345**
		5	43346	43224	43145	43061	43048	**43000**

As it can be observed from the table, GVNS improved 85 out of 250 solutions, when its results are compared to other four algorithms. It was able to obtain 71 equal solutions as other algorithms did. These results are further analyzed in Table 6 in terms of the number of improvements, number of equal and number of worse solutions of GVNS when it is compared to each competing algorithm.

Table 6. Comparison of GVNS with competing algorithms

Algorithm Name	Number of Improvement	Number of Equal Solutions	Number of Worse Solutions
Best	193	47	10
HDDE	184	47	19
VIG_DE	110	66	74
IG_RIS	151	56	43
VIG_FL	127	67	56

In (Tasgetiren M.F., Pan Q., Suganthan P.N., Buyukdagli O., 2013), also the best known solutions for the total flowtime criterion are presented. In order to compare GVNS algorithm's performance in more detailed way, the total flowtime values are provided and compared with the (Tasgetiren M.F., Pan Q., Suganthan P.N., Buyukdagli O., 2013) solutions. For both solutions, the maximum run time is taken as, $T_{max} = n(m/2) \times t$ milliseconds where $t = 60$. In Table 7, the results of each algorithm are represented in the corresponding column. Instances are indicated as, *x_y_z* where *x* is the number of jobs, *y* is the number of machines and *z* is the instance number.

Table 7. Solutions obtained under total flow time criterion

Instance	VIG_DE	GVNS	Instance	VIG_DE	GVNS
50_10_1	125380	125026	300_10_1	2733950	2713771
50_10_2	131385	131117	300_10_2	2707337	2677669
50_10_3	106832	105814	300_10_3	3103430	3072335
50_10_4	101154	101805	300_10_4	2723606	2681871
50_10_5	107635	107697	300_10_5	2884897	2832388
50_20_1	223411	222529	300_20_1	3392341	3321959
50_20_2	224312	224012	300_20_2	4025018	3977650
50_20_3	226144	225245	300_20_3	3667443	3647064
50_20_4	227380	226150	300_20_4	3563901	3541882
50_20_5	178918	178707	300_20_5	3794552	3759573
50_30_1	298825	299472	300_30_1	5417342	5347946
50_30_2	298813	296834	300_30_2	4857488	4811351
50_30_3	292663	291845	300_30_3	5076261	5031066
50_30_4	319430	318373	300_30_4	4822848	4798993
50_30_5	299626	298164	300_30_5	4429028	4382620
50_40_1	395014	392283	300_40_1	5670453	5582637
50_40_2	444201	445392	300_40_2	6349566	6295754
50_40_3	416128	414220	300_40_3	5389312	5249852
50_40_4	405692	404922	300_40_4	6088014	5985650
50_40_5	389638	389210	300_40_5	6431843	6365812
50_50_1	512272	508814	300_50_1	7187544	7174223

Instance	VIG_DE	GVNS	Instance	VIG_DE	GVNS
50_50_2	475389	472957	300_50_2	6705587	6578671
50_50_3	493319	493563	300_50_3	6966348	6858634
50_50_4	429130	429085	300_50_4	7339019	7152297
50_50_5	507519	507881	300_50_5	6557870	6496540
100_10_1	371774	366981	350_10_1	3512173	3483122
100_10_2	343116	340275	350_10_2	4264790	4219816
100_10_3	384173	382838	350_10_3	3838239	3802096
100_10_4	338206	336136	350_10_4	4285421	4254393
100_10_5	376015	375439	350_10_5	3807928	3784085
100_20_1	587306	583452	350_20_1	5523837	5443896
100_20_2	572831	568325	350_20_2	5989808	5954925
100_20_3	624842	623000	350_20_3	4776536	4742525
100_20_4	613650	611482	350_20_4	4959059	4930389
100_20_5	652339	650159	350_20_5	4842744	4803664
100_30_1	871063	862679	350_30_1	5979224	5973182
100_30_2	804380	801285	350_30_2	6507562	6478620
100_30_3	789281	782342	350_30_3	6469955	6410434
100_30_4	836288	833336	350_30_4	7015586	6931007
100_30_5	822142	810814	350_30_5	5800519	5786951
100_40_1	1015413	1002334	350_40_1	6954417	6848557
100_40_2	1046051	1037251	350_40_2	7141816	7104329
100_40_3	989818	990844	350_40_3	9289344	9255174
100_40_4	945786	936833	350_40_4	9052676	9050309
100_40_5	1053204	1050558	350_40_5	7352826	7231418
100_50_1	1328844	1311898	350_50_1	8367007	8235041
100_50_2	1231619	1223158	350_50_2	8674513	8556589
100_50_3	1486911	1480802	350_50_3	9133211	9128903
100_50_4	1411469	1407128	350_50_4	9898786	9870169
100_50_5	1241533	1227216	350_50_5	9275077	9184272
150_10_1	895229	889325	400_10_1	5391735	5336429
150_10_2	748122	741828	400_10_2	4599658	4517166
150_10_3	766768	757007	400_10_3	5166180	5118178
150_10_4	870653	861657	400_10_4	4928345	4880977
150_10_5	855290	852469	400_10_5	4511915	4460947
150_20_1	1066328	1054782	400_20_1	6702405	6664032
150_20_2	1103487	1089865	400_20_2	7101551	7053722
150_20_3	1175110	1162497	400_20_3	6275051	6262191
150_20_4	1068552	1046514	400_20_4	5845455	5764795
150_20_5	1390521	1380852	400_20_5	5836249	5839567
150_30_1	1617787	1598141	400_30_1	7765965	7817646
150_30_2	1474905	1463178	400_30_2	7412412	7368701
150_30_3	1647655	1631426	400_30_3	7214769	7120824
150_30_4	1575960	1562657	400_30_4	8505084	8491659
150_30_5	1640144	1624253	400_30_5	9358018	9258822
150_40_1	1856681	1855503	400_40_1	10803267	10806716
150_40_2	2050045	2028323	400_40_2	9479914	9431776
150_40_3	1896527	1899126	400_40_3	9788011	9796524
150_40_4	1682298	1661276	400_40_4	9861591	9807235
150_40_5	2026015	2013379	400_40_5	9088311	9010777

Instance	VIG_DE	GVNS	Instance	VIG_DE	GVNS
150_50_1	2516185	2497923	400_50_1	11166074	11108239
150_50_2	2300230	2279871	400_50_2	11237488	11159647
150_50_3	2329927	2315911	400_50_3	10887948	10691426
150_50_4	2449533	2436737	400_50_4	12045944	11921693
150_50_5	2354078	2329534	400_50_5	10494339	10424610
200_10_1	1358622	1347340	450_10_1	5533817	5451458
200_10_2	1377316	1364514	450_10_2	6131390	6076454
200_10_3	1404810	1389642	450_10_3	5974770	5886567
200_10_4	1366985	1345497	450_10_4	6647632	6579005
200_10_5	1339260	1341177	450_10_5	5673018	5657365
200_20_1	1865167	1835590	450_20_1	7248737	7118553
200_20_2	1720128	1694132	450_20_2	7356184	7308989
200_20_3	2094286	2077777	450_20_3	7860688	7803450
200_20_4	2152333	2125580	450_20_4	7518296	7471249
200_20_5	1840898	1800455	450_20_5	7474315	7393563
200_30_1	2392036	2378906	450_30_1	10270198	10195381
200_30_2	2321316	2292998	450_30_2	9209464	8996567
200_30_3	2422092	2376247	450_30_3	9397035	9278882
200_30_4	2863910	2826109	450_30_4	9790031	9689365
200_30_5	2511521	2478473	450_30_5	9450992	9205719
200_40_1	3020448	3006439	450_40_1	12610909	12500237
200_40_2	3182705	3152290	450_40_2	11153309	10973657
200_40_3	3144740	3120945	450_40_3	11798619	11695071
200_40_4	2550159	2507863	450_40_4	11798390	11644774
200_40_5	3179470	3146713	450_40_5	10912158	10887270
200_50_1	3535086	3508156	450_50_1	12011731	11808223
200_50_2	3753945	3724370	450_50_2	14690680	14677991
200_50_3	3469069	3396039	450_50_3	14564560	14397517
200_50_4	3758949	3763278	450_50_4	13652052	13500037
200_50_5	3793772	3763148	450_50_5	13590028	13523934
250_10_1	2385078	2360814	500_10_1	7598974	7522649
250_10_2	2106167	2066518	500_10_2	7102394	7022973
250_10_3	2036196	1999403	500_10_3	6957682	6882112
250_10_4	2016989	1988889	500_10_4	7062995	7015011
250_10_5	1966408	1945311	500_10_5	7290714	7220547
250_20_1	2799970	2769381	500_20_1	10777501	10717624
250_20_2	2699738	2671260	500_20_2	9682988	9652758
250_20_3	2704526	2662172	500_20_3	9058512	9000096
250_20_4	2765178	2747653	500_20_4	8726791	8706843
250_20_5	2655471	2641649	500_20_5	10223564	10191122
250_30_1	3864142	3844038	500_30_1	11648780	11602319
250_30_2	3542779	3489081	500_30_2	12888556	12839912
250_30_3	3428481	3369936	500_30_3	13051895	13024061
250_30_4	3370868	3322982	500_30_4	10403033	10287342
250_30_5	3702931	3677503	500_30_5	12323304	12242723
250_40_1	4158216	4064573	500_40_1	14019665	13937648
250_40_2	4271224	4177475	500_40_2	15153155	15128277
250_40_3	4457779	4438585	500_40_3	13706925	13538076
250_40_4	4580109	4534462	500_40_4	14577131	14320458

Instance	VIG_DE	GVNS	Instance	VIG_DE	GVNS
250_40_5	4243709	4158624	500_40_5	11902207	11605651
250_50_1	5570902	5534031	500_50_1	16633031	16496225
250_50_2	4724396	4591831	500_50_2	15315588	15251117
250_50_3	5150808	5097909	500_50_3	16252743	16191002
250_50_4	4808325	4702710	500_50_4	15497661	15245952
250_50_5	5166847	5112169	500_50_5	15283675	15169049

As it can be observed from Table 7, GVNS improved 236 out of 250 current best known solutions under the total flow time criterion obtained by using the VIG_DE algorithm that is proposed in (Tasgetiren M.F., Pan Q., Suganthan P.N., Buyukdagli O., 2013), as well.

CHAPTER 5

CONLUSION

In this thesis study, a metaheuristic algorithm for no idle permutation flowshop problem is represented. In a no idle permutation flowshop, there is more than one machine and each job must be processed on each of the machines. Each job has the same ordering of machines for its process sequence and the processing sequences of the jobs are the same on each machine. Each job can be processed on one machine at a time, and each machine can process only one job at a time, each machine must process each job without any interruption from the beginning of the first job to the completion of the last job. Two different formulations that are proposed to calculate makespan of no-idle flowshop are explained and the one with the better performance is selected.

After a detailed explanation of the problem, first the competing algorithms that are proposed in the literature to solve NIPFS problem is explained. These algorithms are; (1) an iterated greedy, (2) variable iterated greedy, (3) the hybrid discrete differential evolution and (4) variable iterated greedy algorithm with differential evolution algorithm. Then, a new version of the GVNS algorithm with insert and swap operations in outer loop, and in the inner loop, IG algorithm and iterated local search algorithm is applied to NIPFS problem and then compared to all other algorithms. The performances of the proposed algorithms are tested on the Prof. Ruben Ruiz's benchmark suite.

Computational results are proposed and concluded as the GVNS algorithm further improved 85 out of 250 current best known solutions. This high performance of the algorithm can be explained as; using that much powerful algorithm as a neighborhood structure instead of more basic ones, increases this phase's ability to reach better solutions. In addition, these conclusions are supported by the paired T-tests and the interval plot.

Bibliography

Adiri, I. and Pohoryles, D. Flow-shop/no-idle or no-wait scheduling to minimize the sum of completion times [Journal]. - [s.l.] : Naval Research Logistics Quarterly, 1982. - 3 : Vol. 29. - pp. 495–504..

Baptiste, P., & Lee, K. H. A branch and bound algorithm for the F|no-idle|Cmax [Journal]. - Lyon : Proceedings of the international conference on industrial engineering and production management (IEPM), 1997. - Vol. 1. - pp. 429–438.

Deng G and Gu X. A hybrid discrete differential evolution algorithm for the no-idle permutation flowshop scheduling problem with makespan criterion [Journal]. - [s.l.] : Computers & Operations Research, 2012. - 9 : Vol. 39. - pp. 2152-2160 .

Framinan and Leisten Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm [Journal]. - [s.l.] : Int. J. Prod. Res., 2008. - 22 : Vol. 46. - pp. 6479 - 6498.

H. Saadani, A. Guinet, M. Moalla Three stage no-idle flow-shops [Journal] // Computers and Industrial Engineering. - 2003. - Vol. 44. - pp. 425–434.

H.R. Lourenc, O. Martin, T. Stützle Iterated local search [Book Section] // Handbook of Metaheuristics, International Series in Operations Research & Management Science / book auth. F. Glover G. Kochenberger (Eds.). - Norwell, MA : Kluwer Academic Publishers, 2002. - Vol. 57.

Hansen P., Mladenovic N., Urosevic D. Variable neighborhood search and the local search and local branching [Journal]. - [s.l.] : Computers & Operations Research, 2006. - 10 : Vol. 33. - pp. 3034-3045.

Kalczynski, P.J. and Kamburowski, J. On no-wait and no-idle flow shops with makespan criterion [Journal]. - [s.l.] : European Journal of Operational Research, 2007. - 3 : Vol. 178. - pp. 677-685.

Kirlik G., Oguz C. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine

[Journal]. - [s.l.] : Computers & Operations Research, 2012. - Vol. 39. - pp. 1506-1520.

Mladenovic´, N., Hansen, P. Variable neighborhood search [Journal]. - [s.l.] : Computers and Operations Research, 1997. - Vol. 24. - pp. 1097–1100.

Nawaz, M., Enscore, Jr, E. E., and Ham, I. A heuristic algorithm for the m-machine, njob flow-shop sequencing problem [Journal]. - [s.l.] : OMEGA, The International Journal of Management Science, 1983. - 1 : Vol. 11. - pp. 91–95..

Pan, Q-K. and Wang, L. A novel differential evolution algorithm for no-idle permutation flowshop scheduling problems [Journal]. - [s.l.] : European Journal of Industrial Engineering, 2008. - 3 : Vol. 2. - pp. 279–297.

Ruiz R., Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem [Journal]. - [s.l.] : European Journal of Operational Research, 2007. - 3 : Vol. 177. - pp. 2033-49.

Ruiz, R. , Vallada, E. , Fernández-Martínez, C. Scheduling in Flowshops with No-Idle Machines [Journal]. - [s.l.] : Computational Intelligence in Flow Shop and Job Shop Scheduling Studies in Computational Intelligence, 2009. - Vol. 230. - pp. 21-51.

Ruiz, R., Vallada, E., and Fernandez-Martinez, C. Scheduling in flowshops with no-idle machines [Journal] // Computational Intelligence in Flowshop and Job Shop Scheduling / ed. Chakraborthy U.K. - Berlin : Heidelberg: Springer Verlag, 2009.

Saadani, N. E. H., Guinet, A., and Moalla, M. A travelling salesman approach to solve the F/no−idle/Cmax problem [Journal]. - [s.l.] : European Journal of Operational Research, 2005. - 1 : Vol. 161. - pp. 11–20.

Storn R, Price K. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Space [Journal]. - [s.l.] : Journal of Global Optimization, 1997. - Vol. 11. - pp. 341-359.

Stützle Thomas Local Search Algorithms for Combinatorial Problems—Analysis, Algorithms and New Applications [Book]. - Sankt Augustin, Germany : [s.n.], 1999.

Taillard E. Some efficient heuristic methods for the flow shop sequencing problem [Journal]. - [s.l.] : European Journal of Operational Research, 1990. - 1 : Vol. 47. - pp. 65-74.

Tasgetiren M.F., Pan Q., Suganthan P.G., Oner A. A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion [Journal]. - [s.l.] : Applied Mathematical Modelling, (In Press).

Tasgetiren M.F., Pan Q., Suganthan P.N., Buyukdagli O. A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem [Journal]. - [s.l.] : Computers & Operations Research, 2013. - Vol. 40. - pp. 1729–1743.

Tasgetiren M.F., Pan Q.K., Suganthan P.N., Liang Y.C. A discrete differential evolution algorithm for the no-wait flowshop scheduling problem with total flowtime criterion [Journal]. - Hawaii, USA : Proceedings of the 2007 IEEE symposium on computatiıonal intelligence in scheduling., 2007b. - pp. 251-258.

Tasgetiren, M. F., Pan, Q. -K., Liang, Y. -C., Suganthan, P.N. A discrete differential evolution algorithm for the total earliness and tardiness penalties with a common due date on a single machine [Journal]. - Hawaii : In Proceedings of the 2007 IEEE symposium on computational intelligence in scheduling (CISched2007),, 2007a. - pp. 271–278.

Thomas Stützle Iterated local search for the quadratic assignment problem [Journal]. - [s.l.] : European Journal of Operational Research, 2006. - Vol. 174. - pp. 1519-1539.

Vachajitpan P. Job sequencing with continuous machine operation [Journal]. - [s.l.] : Computers and Industrial Engineering, 1982. - 3 : Vol. 6. - pp. 255-259.

Wolpert D. H. and Macready W. G. No Free Lunch Theorems for Optimization [Journal]. - [s.l.] : IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 1997. - Vol. 1.

Woollam C. R. Flowshop with no idle machine time allowed [Journal]. - [s.l.] : Computers and Industrial Engineering, 1986. - 1 : Vol. 10. - pp. 69–76.