HYBRID CONTINUOUS SCATTER SEARCH APPROACH TO TARDINESS RELATED SCHEDULING PROBLEMS

by

Ercüment Erdur

B.S., Mechanical Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2006

To my mother, Muhsine Hanım...

ACKNOWLEDGEMENT

First and foremost, I am grateful to my thesis supervisor Prof. Ümit Bilge for privileging me to work with her, and for her continuous guidance and support through this study. It was really a great pleasure for me to work with her.

I would like to express my gratitude to Prof. Tülin Aktin and Assoc. Prof. Necati Aras for taking the time to examine this thesis, and taking part in my thesis jury.

My deepest gratitude goes to my family: my mother, my father, my brothers; Emir and Enver Erdur, my aunt Adnan Akyazıcı for their never ending love, support and trust, not only during this study, but also through my life.

I would thank to my friends Özhan Özbe, Billur Kılınç, and Ali Kemal Karagöz for sacrificing numerous hours to help typing my thesis. Also, I wish to thank Birkan Demirci, Soner Arıcı, Aslıhan Akel, Güler Kızılcık, Hande Çıtakoğlu and Nihan Karali for their continuous support and motivation, which stop me to feel alone when I lost my way throughout the hard times of my thesis.

ABSTRACT

HYBRID CONTINUOUS SCATTER SEARCH APPROACH TO TARDINESS RELATED SCHEDULING PROBLEMS

In this thesis, a hybrid approach, which integrates Scatter Search (SS) and a Variable Neighborhood Search (VNS), is presented to attack tardiness related scheduling problems. The aim is to find advanced strategies that can be adapted to the basic SS methodology in order to enhance its diversification and intensification capabilities throughout the scheduling problems. The Hybrid Continuous Scatter Search (HCSS) approach is first implemented on the Single Machine Total Weighted Tardiness (SMTWT) problem to minimize total weighted tardiness. Then the HCSS method is modified to addresses the Parallel Machine Total Tardiness (PMTT) problem, which consists of a set of jobs to be scheduled on a number of parallel processors to minimize total tardiness. The NP-hard nature of both problems renders a challenging area for research.

In order to develop a robust hybrid methodology, the key elements of the Scatter Search such as reference set update method, initial solution generation method, solution combination method and as an intensification strategy – the hybridized VNS are investigated. The employed solution encoding, diverse solution selection methods, and dynamic solution combination method are unique and introduced first time in this thesis to provide new ideas for Scatter Search era. The proposed HCSS approach yields good quality results with respect to optimal/best-known solutions reported in the literature.

ÖZET

ARTI GECİKME TABANLI ÇİZELGELEME PROBLEMLERİNE MELEZ SÜREKLİ DAĞILIM ARAMASI YAKLAŞIMI

Bu tezin konusu olan çalışmada, Artı Gecikme Tabanlı Atama problemlerini çözmek için Dağılım Araması (DA) ve Değişken Komşuluk Araması (DKA) yöntemlerini bünyesinde birleştiren melez bir yaklaşım sunulmuştur. Bu tezde amaç, DA metodunun atama problemlerinin çözümündeki başkalaşım ve kuvvetlendirme kabiliyetlerini artırmak için temel metodolojisine adapte edilebilecek ileri seviye stratejiler bulmaktır. Melez Sürekli Dağılım Araması (MSDA) yaklaşımı ilk olarak Tek Makina Toplam Ağırlıklı Artı Gecikme (TMTAG) probleminde toplam ağırlıklı artı gecikmeyi en küçüklemek için yürütülmüştür. Bir sonraki kademede MSDA metodu, bir takım işin birkaç paralel işlemci üzerinde toplam artı gecikmesini en küçüklemek amacıyla oluşturulan Paralel Makina Toplam Artı Gecikme (PMTAG) problemini ele alabilmesi için uygun bir şekilde modifiye edilmiştir. İlgilenilen problemlerinin NP-zor doğası itibari ile ortaya iddialı bir araştırma konusu çıkmıştır.

Sağlam bir metodoloji geliştirmek için Dağılım Araması yönteminin anahtar elemanları olan; referans kümesi güncelleme metodu, başlangıç çözümü oluşturma metodu, çözüm birleştirme metodu ve çözüm kuvvetlendirme stratejisi olarak DKA yöntemi incelenmiştir. Kullanılan çözüm kodlaması, farklı çözüm seçme metodu ve dinamik çözüm birleştirme metodu DA yöntemine yeni fikirler teşkil etmek amacı ile ilk defa bu tezde sunulmuşlardır. Tasarlanan MSDA yaklaşımı literatürde yayınlanmış bilinen en iyi çözümlerle karşılaştırıldığında kaliteli sonuçlar vermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

b_1	Number of elite solutions
b_2	Number of diverse solutions
C	Completion time
d	Due date
m	Number of machines
$N( )$	Neighbors of a current solution
n	Number of jobs
p	Processing time
r	Ready time
$rand$	Random number
s	Setup time
S	Slack time
st	Start time
T	Tardiness
π	Priority value
Δ	Increment value
ρ	Average utilization of machines
ACO	Ant colony optimization
B&B	Branch and bound
CBN	Combination rule
CPU	Central processing unit
EA	Evolutionary algorithm
EDD	Earliest due date
EFT	Earliest finish time
EJOR	European Journal of Operational Research
FVNS	Final variable neighborhood search
GA	Genetic algorithm
HCSS	Hybrid continuous scatter search

LB	Local branching
MA	Memetic algorithm
MVNS	Middle variable neighborhood search
NJS	Neighborhood job swap
NP	Non-deterministic polynomial
OR	Operation research
PMTT	Parallel machine total tardiness
PSO	Particle swarm optimization
R&M	Rachamadugu and Morton
RDD	Range of due dates
RVNS	Reduced variable neighborhood search
SA	Simulated annealing
SMTWT	Single machine total weighted tardiness
SPT	Shortest processing time
SS	Scatter search
TF	Tardiness factor
TS	Tabu search
TWLW	Total weighted late work
twt	Total weighted tardiness
VND	Variable neighborhood descent
VNDS	Variable neighborhood decomposition search
VNS	Variable neighborhood search

1. INTRODUCTION

Designing efficient solution algorithms for Combinatorial Optimization problems, especially scheduling problems, has been a real challenge for researchers. Scheduling problems arise in many contexts; from computer engineering to manufacturing techniques. Most scheduling problems are NP-hard; thus render classical approaches such as Branch and Bound schemes or integer linear programming unpractical for real-life instances. Due to the fact that the quality of simple approximation approaches remains limited, many researchers continuously search for new sophisticated methods which can provide solutions with high accuracy in a short time.

Meta-heuristics, which extend the neighborhood search beyond the local optima, introduced a new era towards improvement of solution quality for scheduling problems. Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GA) are the most established and widely used meta-heuristic approaches. Several studies in the literature report successful implementations of these methods on scheduling problems where the TS and its hybrids were often surpassing the performance of the others.

Scatter Search (SS) is another population-based meta-heuristic that came into attention very recently, although the original idea dates back to 1970s. Basically, SS works on a set of good but diverse reference solutions to generate new trial solutions by combining them. European Journal of Operational Research published a special volume (EJOR Vol. 169, 2006) dedicated to SS and its implementations. In this volume, articles on methodology of SS, as well as several applications including assignment, routing, clustering and scheduling problems are reported. In his editorial article, Marti (2006) states the highlights of SS and presents it as an amazing search method which would break the dominance of Tabu Search and Genetic Algorithm. Based on Marti's analysis, the number of SS publications shows a boost after year 2000. It is apparent that this approach is still not fully mature, needs further development and testing especially for scheduling domain.

The highly concentrated interest on SS has motivated and encouraged us to test this new approach in this thesis. Due to its stochastic nature and the absence of an exploitation

component, SS might result poorly when the NP-Hard scheduling problems are taken into consideration. Therefore, SS approach is enhanced with a well known local improvement method namely Variable Neighborhood Search (VNS) whose performance on intensification is proven throughout most of the studies performed in scheduling era.

In order to test the performance of the SS approach, we selected two tardiness related scheduling problems. Tardiness, or the duration by which the completion time of a job exceeds its due date, is a challenging performance measure in the sense that even the simplest scheduling setting of sequencing *n* jobs on a single machine becomes NP-hard under this objective. Moreover, scheduling against due dates receives considerable attention in literature, since delivery time performance becomes an increasingly critical issue under the growing pressure of the competition in today's markets.

The first problem handled is minimizing the total weighted tardiness of *n* jobs scheduled on a single machine (SMTWT), a well known NP-hard problem. The problem has been studied previously by several researchers including: TS, SA, and GA approaches by Crauwels et al. (1998), dynasearch algorithm by Congram et al. (2002) and TS approach by Bilge et al. (2007). There is a set of benchmark problems whose best known solutions are largely provided by Congram, and TS algorithm introduced by Bilge et al. produces matchingly good results.

The second problem selected is the Parallel Machine Total Tardiness (PMTT) problem. Here, a set of jobs with distinct arrival times and sequence dependent setup times are scheduled on a set of uniform parallel machines to minimize total tardiness. This complicated and realistic problem has also been studied by various researchers: GA approach by Sivrikaya-Şerifoğlu and Ulusoy (1999), TS by Bilge et al. (2004), GA by Bilge and Kıraç (2006) and a hybrid of TS-SA-VNS by Anghinolfi and Paolucci (2006). The best-known solutions for the benchmark set of problems, -first introduced by Sivrikaya-Şerifoğlu, has been improved by each research in this line, and currently the results obtained by Anghinolfi and Paolucci constitute the best-known solutions.

In this thesis, the proposed hybrid SS will be tested over these two benchmark problems sets and the results will be compared to the most up-to-date best known

solutions. The thesis is organized as follows: In the next chapter, after a brief overview of heuristic approaches in general and a meta-heuristics in particular, a more through account for both SS and VNS are given. Chapter Three provides detailed problem definitions and literature surveys for the SMTWT and PMTT problems. The proposed hybrid SS algorithm is described in details in Chapter Four. Numerical studies and final results for both problem settings are presented in Chapter Five and finally, Chapter Six ends the thesis with summary and conclusive remarks.

2. OVERVIEW OF META-HEURISTICS AND SCATTER SEARCH

The hybrid approach used in this study, integrates two well-known meta-heuristics; Scatter Search and Variable Neighborhood Search. Thus it will be a good idea to start with basics and define heuristics and meta-heuristics in general before examining our approach.

The term heuristic means a method based on previous experiences and judgments that provides a fast and reasonable solution to a problem, but which cannot be guaranteed to produce the mathematically optimal solution (Silver, 2004). There are several types of heuristic solution methods that can be chosen defining on factors related to the problem. Most common heuristic techniques can be categorized as constructive methods, branch and bound derivatives and local improvement (neighborhood search) methods. Constructive methods as the name implies use the data of the problem to build a solution step be step. Hence no solution is obtained until the procedure is complete. At each iteration, there is a partial solution and the extension of the current solution is constructed by selecting one of the possible options available for the current status of the solution. This makes the approach myopic. Usually the option with the minimum cost is selected as the extension leading to so called greedy method. As an example, consider a scheduling problem concerned with the sum of the weighted completion times of n jobs on a single machine. Job j has a process time p_j and a weight w_j. For this case, a well known dispatching rule so called WSPT can be employed as a constructive algorithm. All the jobs are ranked according to their w_j / p_j ratios in decreasing order and whenever the machine is idle the job with the highest rank is processed and continues to the next highest until set of unprocessed jobs is empty. (Pinedo and Simchi-Levi, 1996)

Enumerative branch and bound methods are widely used to obtain optimal solutions to NP-hard scheduling problems. Branch and Bound (B&B) attempts to eliminate a node by determining a lower bound on the objective function for all partial schedules that derived from that node. If the lower bound is higher than the value of objective function under known schedule, the node may be eliminated and its possible offspring are neglected. The main advantage of Brach and Bound methods is that after evaluating all

nodes, the final solution can be considered to be the optimal. On the other hand, B&B method is extremely time-consuming, when the number of nodes is very large. Therefore some derivatives of B&B method are constructed to overcome this handicap. Filtered beam search is an adaptation of Branch and Bound method in which only the most promising nodes at a level are selected as nodes for further branching. The remaining nodes at that level are filtered permanently. The number of nodes kept is so called beam width of the search. The decision process that determines the promising nodes is the most important phase of this method. There is a trade-off between quickness and efficiency. A crude prediction is quick, but may lead to discard good solutions. On the other hand, more through evaluations may be extremely time consuming. (Reeves, 1993)

The basic concept behind the local improvement methods is quite simple. One starts with a feasible solution to a problem, often generated randomly or obtained as a result of a constructive method. In the next step, feasible solutions in the neighborhood of the current solution are evaluated. If one of the new solutions is better than the current solution, it becomes the new current solution and its neighborhood is searched for the next iteration. These repetitive steps are continued until no improvement can be found. The current solution, at the final stage is accepted as the local optimum. The neighborhood of the solution suggests that two solutions are neighbors if one can be obtained through well defined modifications. As an example, for a single machine scheduling problem, a solution is a specific sequence of the jobs on a machine and neighborhood of the current solution can be defined as the new schedule obtained by performing a single adjacent pair-wise interchange of two consecutive jobs. While searching the neighborhood of the current solution, an important issue arises. It is crucial to decide whether to choose a move to the first solution in the neighborhood exhibiting an improvement or to evaluate all the solutions in the neighborhood and choose the one giving the largest improvement. (This method is often referred to as steepest ascent (or descent) method)(Reeves, 1993). Depending on the new current solution selection strategy, different local improvement methods can be constructed. Although these methods perform highly satisfactory, they only guarantee a local optimum. The final solution obtained heavily depends on the starting solution and most possibly ends at a local optimum. In order to break out the fundamental weakness of local search, exploration and diversification methods are

suggested to broaden search to other parts of the solution space. The resulting heuristics are generally named as meta-heuristics.

2.1. Meta-heuristics

A meta-heuristic is an iterative master process that guides and modifies the operations of sub ordinate heuristic to produce efficiently high quality solutions. It may combine intelligently different concepts to explore solution space using adaptive learning strategies and structured information (Osman, 1996). Meta-heuristics are particularly concerned with not getting trapped at local optimum and carefully reducing solution space to be searched. Every meta-heuristic has one or more adjustable parameters, that provide flexibility and the robustness. On the other hand, this parameter requires intensive calibration on set of numerical instances of the problem. The family of meta-heuristics can be classified into three main categories and their sub-categories. Construction based meta-heuristics includes greedy random adaptive search methods, and guided construction methods. Local search based meta-heuristics include simulated annealing, noise methods, guide local search methods, iterated local search, neural networks, Tabu Search and variable neighborhood search. Population based meta-heuristics include evolutionary algorithms (EA) such as ant colony systems, particle swarm optimization, genetic algorithm, and scatter search. (Osman and Kelly, 1996)

Tabu search (TS) is one of the most widely used meta-heuristic proposed by Glover (1997). In his paper, Glover points to several application areas including scheduling, routing, location/allocation, design, logic and artificial intelligence. TS begins with a complete feasible solution. This solution can be either randomly generated one or a more qualified solution evaluated by a constructive heuristic. Just like the other local improvement methods, TS continues to develop new complete solutions from its neighborhood. Then these candidate solutions are evaluated and chosen if better than the current solution. However, evaluating every possible move from the current solution might be extremely time consuming and computationally expensive. Therefore, candidate list strategy is employed to filter some neighborhood solutions. A candidate list strategy chooses potentially good candidates from the neighborhood, and prevents considering all moves. In order to avoid being trapped at local optimum, moves to neighborhood points

with inferior solutions are permitted. In addition to that a mechanism is used to prevent cycling back to recently visited solutions. Recently visited solutions are kept in a tabu list and these solutions are avoided from reoccurring for a certain number of iterations. The number of tabu iterations also called tabu tenure is a key controllable parameter of TS.

Two crucial components of TS search are intensification and diversification strategies. Intensification strategy is based on modifying choice rules to encourage move combinations and solution features historically found good whereas diversification strategy encourages the search to examine unvisited regions of search space and to generate solution that differs significantly than previously visited. The memory used in TS is both explicit and attributive. Explicit memory records elite solutions visited during the search and extension of this memory stores highly attractive but unexplored neighborhoods of elite solutions. These solutions are later used to expand local search.

Simulated Annealing (SA) takes its name from the physical process called annealing where a material is heated into a liquid state then cooled back into recrystallized solid state. In this local search method, the randomly generated or a constructively evaluated solution is considered at the initial stage. This starting solution must be complete and feasible. In the following steps, neighborhood solutions are generated by mutations and evolutions. If the candidate solution is better than the current one, it becomes the new current solution. However, if the fitness of candidate is inferior to the current, then there is still a chance that candidate replaces current with a probability determined by exponential of the difference between fitness values and a parameter called the temperature. The probability of accepting the poor solution decreases as the difference between fitness values increases or the temperature becomes smaller. The temperature is gradually lowered during the course of the search so that the probability of accepting poor solutions is reduced towards the end of the process. The search continues until the termination criterion is met. This criterion can be a certain total number of iterations or a prescribed number of consecutive iterations without any improvement. These properties of simulated annealing are more or less same as the TS. However, there are important differences between the methods. Firstly, TS uses adaptive memory, whereas SA is memoryless. Secondly, TS tends to permit moving to an inferior solution when in the vicinity of the local optimum whereas this can happen at any time in SA. Finally, TS allows moving away from a local

optimum by a deterministic mechanism; on the other hand SA uses a probabilistic method. (Jones and Rabelo, 1998)

Ant Colony Optimization (ACO) proposed by Dorigo (Dorigo and Di Caro, 1999) is one of the most successful examples in swarm intelligence systems and have been applied to many types of optimization problem. This algorithm simulates the behavior of ant colonies when finding the shortest path between nest and the food source. While going from nest to food source (or vice versa), ants deposit a chemical substance called pheromone on the ground. When they arrive the decision point such as an intersection between shorter and longer branch, they make a choice depend on the amount of pheromone they smell on the two branches. For this reason, ants choose the shorter path having more pheromone with higher probability then the longer one. New pheromone is released on the chosen path and makes it more attractive for subsequent ants. Consider a traveling salesman problem; the probability of a salesman transition from one city to another depends on two factors. First one is the direct distance between two cities (probability that is inversely proportional to distance) and the second one is the remaining amount of pheromone released by earlier salesman that has traveled this link (probability that is proportional to amount of pheromone). In order to provide flexibility and prevent premature convergence, ant colony optimization method uses a predefined formula known as pheromone evaporation rate through which remaining pheromone is updated at the end of each iteration.

Particle Swarm Optimization (PSO), introduced by Kennedy and Eberhart (2001), is based on a social-psychological model of social influence and social learning. Individuals in a particle swarm follow a very simple behavior: emulate the success of neighboring individuals. The collective behavior, which emerges, is that of discovering optimal regions of a high dimensional search space. The swarm of particles responds to quality factors in the form of the personal and neighborhood best positions. Allocation of responses between the personal best and neighborhood best positions ensure a diversity of response. The particle changes its state only when the personal best and the global best position change. A PSO algorithm maintains a swarm of particles, where each particle's position represents a potential solution. In analogy with evolutionary computation paradigms, a swarm is similar to a population, while a particle is similar to an individual.

In simple terms, the particles are flown through a multi-dimensional space, where the position of each particle is adjusted according to its own experience and that of its neighbors. The position of the particle is altered by adding a velocity to the current position. This velocity vector drives the optimization process. At the end of search, the particle with the superior position is considered as the best particle of the swarm and represents the best found solution.

Genetic Algorithms (GA) (Goldberg, 1989, Liepins and Hillard, 1989) work with a group or population of solutions. Each individual in the population is characterized by its fitness. The fitness of an individual is associated to the objective function. The process works iteratively and each iteration is referred to a generation. A generation consists of surviving individuals and new solutions or children from previous generation. Population size generally remains constant from one generation to the next. The children are generated through reproduction and mutation of individuals that are part of previous generation. At each iteration, the fittest individuals reproduce and the least fits die. The birth, death and reproduction procedure that determine the composition of the next generation can be complicated process that is usually a function of the fitness levels of the individuals of the current generation. There are considerable number of controllable parameters and other choices including size of population, the probability of mutation of an individual, and the number of crossover points, etc.

Scatter Search is another derivative of evolutionary algorithms that uses weighted linear combinations of several solutions contained in a reference set to produce new solutions. Since we employ Scatter Search methodology in our hybrid approach, detailed definitions of SS internal components and their working mechanisms are given in the next section.

2.2. Scatter Search

Scatter Search (SS) is an evolutionary algorithm which was first introduced by Glover (1977) as a heuristic for integer programming. In contrast to other evolutionary methods like GA, SS is founded on the idea that new solutions created by systematic design and procedures provide significant benefits than those generated randomly. SS

orients its exploration systematically relative to a set of reference points that typically consists of good solutions obtained by prior problem solving efforts. In this context, reference solutions do not cite as "good" according to their objective function value, may also be considered in the case of some specifications that differ from other solutions. SS uses weighted linear combinations to produce new candidate solutions within the search space containing previously found reference points.

Glover (1998) provides a Scatter Search template as the main reference for most of the SS implementations. He defines five components to construct a basic design as follows:

- A diversification method to generate a collection of diverse trial solutions using an arbitrary trial solution as an input.
- An improvement method to transform a trial solution into a more enhanced trial solution
- A reference set update method to build and maintain a reference set containing predefined number of best solutions and diverse solutions which are accepted according to their quality or diversity.
- A subset generation method to operate on reference set to produce a subset of its solutions as a basis for creating combined solutions.
- A solution combination method to transform a given subset of solutions produced by subset generation method into one or more combined vectors.

The reference set plays the crucial role in SS method. The reference set (*RefSet*) as mentioned previously is a collection of both high quality and diverse solutions that are used to generate new candidate solutions. The number of elite solutions b_1 and diverse solutions b_2 are fixed and state the size of the reference set $(b = b_1 + b_2)$. The construction of the initial reference set starts with the selection of the best b_1 solution from the pool of initially generated and improved trial solutions. These solutions are added to *RefSet* and discarded from the pool. For each solution in the pool, the minimum of the distances to the reference points contained in *RefSet* is computed. Then the solution with the maximum of these minimum distances is selected as the diverse point. This solution is added to *RefSet* and deleted from the pool. The loop continues until b_2 diverse solutions are chosen to *RefSet*. (Laguna et al., 2006)

After constructing the *RefSet*, subsets are created by using the subset generation method. In traditional EA's such as GA, parents are selected through a random sampling scheme, but in SS the parent selection is based on a deterministic method. This method generates all subsets of size two while skipping subsets with the same elements. Using an appropriate solution combination method, these subsets yield into new trial solutions that build up the pool. Then the *RefSet* is rebuilt using new pool and previous *RefSet* through mentioned update method. These five methods are repeated simultaneously until a termination criterion is met. (Laguna and Marti, 2003)

Some advanced design, such as dynamic *RefSet* updating, *RefSet* rebuilding, *RefSet* tiers, and diversity control are outlined by Glover (1998) which can be useful modifications to improve the performance of SS implementations. In dynamic *RefSet* updating, *RefSet* is updated immediately when a new best solution is found by the combination method. The advantage of dynamic update is that it quickly replaces inferior solutions in the *RefSet* with better solutions and future combinations are made with improved solutions. The disadvantage is that, some potentially promising combinations are eliminated before being considered. In *RefSet* rebuilding method, *RefSet* is partially rebuilt with diversification update where the previous diverse solutions are deleted and new diverse solutions are generated using diversification generation method. These solutions are then placed to *RefSet*. Another advanced approach is dividing *RefSet* into more than two subcategories. As previously mentioned, *RefSet* already consists of two subcategories: high quality and diverse points. In addition to these, new categories may be defined, such as a third category containing good generators which generated high quality trial solutions when used as inputs in the combination method for the previous generations. All these advance modifications can improve the performance of SS and translate it into a higher complexity with additional search parameters. However, it conflicts with the goal of designing a method that is easy to implement and fine tune. Therefore, there is no one and exact combination of these methods that leads to the best performance. One should try and decide which combination is superior for a given problem context.

Initial step for an efficient SS algorithm is to understand its methodology. Glover (1998), Glover et al. (2000), Glover et al. (2003), Greistorfer (2004), Herrera et al. (2006), Laguna et al. (2003), Laguna et al. (2006), Reeves and Yamada (1999) develop basic

mechanisms and methods of SS design. They also provide insights to efficient implementations and advanced modifications that would help while aiming complex problems. The methodologies given in these papers become the keystones of our SS approach which will be discussed briefly.

As described in those papers, SS is implemented for both discrete and continuous optimization problems. Since it is a methodology, the described components of SS can be refined and adapted depending on the nature of the considered optimization problem. A discrete optimization problem consists of integer variables x_i's such that $x_i \in Z$, whereas continuous problem includes real-value x_i's that belongs to the set R. Constructing a proper solution encoding is one of the most crucial parts of the SS approach. Depending on the structure of the problem, encoding might contain either integer values or real-value components, which should provide a meaningful mapping to the search space. The components of SS such as solution combination method and diverse solution selection method are designed according to the utilized solution encoding.

The generation method of the new solution changes according to the type of the encoding. For discrete problems, the solution combination method employs a crossover or a mutation operator which generate a new individual by crossing chromosomes or mutating them. However, real-value encoding of solutions offers the possibility of defining a wide variety of special real-parameter combination operators which can take advantage of its numerical nature. These operators construct intervals depending on the linear combinations of the solutions and select a new solution within this interval. Average combination method and BLX-α are two well-known combination methods for continuous GAs. (Herrera et al., 2006)

Another important method to be carefully designed is the diverse solution selection method. Diverse solutions are chosen with respect to their distance values to the reference set. This distance measurement should be performed in an accurate way in order to select superior diverse solutions. The diversity of two solutions can be defined by either calculating the distance between them or evaluating the dissimilarities inside their structures. The most of the diverse solution selection methods originates from these two measurement rules. Every SS approach has its own unique distance measurement function

depending on the characteristics of the desired diverse solution, and the nature of the solution encoding.

The hybrid applications show that SS can be improved when combined with other meta-heuristics. The key idea behind these hybrid approaches should be identifying the method with required strategy and to justify the selection of this method from another meta-heuristics. Investigating some hybrid methods found on literature assists us to construct our own hybrid SS.

Greistorfer (2003) propose a meta-heuristic based on a TS procedure that makes use of the SS paradigm to solve a capacitated Chinese Postman problem where for a given undirected network in which the goal is to determine a least cost schedule of routes. The computational results indicate that the algorithm can cope up with the other arc routing heuristics.

Nowicki and Smutnicki (2006) provide a new view on the solution space and the search process of flow-shop makespan problem. They present a new approximate algorithm which applies some properties of neighborhood approach known as big valley phenomenon, uses some elements of SS as well as the path-relinking technique. The proposed algorithm provides very good accuracy obtainable in a short CPU time when compared to other best known methods.

Liu (2006) presents a hybrid SS by incorporating the nearest neighbor rule, threshold accepting and edge recombination crossover into a scatter search conceptual framework to solve the probabilistic traveling salesman problem. The author conducts several experiments to test the validity of a hybrid SS on the test problems attained from literature. His numerical analysis proves that incorporating threshold accepting into SS increases the computational efficiency while maintaining solution quality.

Pachebo (2005) states a meta-heuristic algorithm based on SS whose aim is to obtain quality solutions with short computation times for the non-hierarchical clustering problem under the criterion of minimum sum of squares clustering. He combines several

procedures based on different strategies such as local search, GRASP, TS with Scatter Search.

There are more reference papers that are cited by Marti on hybrid SS approaches and on other problem types. All prove that, incorporating SS with other types of meta-heuristics depending on what we might need during our search can overcome the chronic handicaps associated with SS and make it more robust against different problem types. To sum up, hybrid SS approaches provide better and faster results than the SS methods or other meta-heuristics. These results encourage us to develop a hybrid scatter search approach to attack scheduling problems considered in this thesis. The next section briefly overviews the Variable Neighborhood Search method which will be used as an intensification strategy in our hybrid approach.

2.3. Variable Neighborhood Search

Variable neighborhood search (VNS) is a systematic change of neighborhood within a possibly randomized local search algorithm that yields a simple and effective meta-heuristic for combinatorial and global optimization problems. Contrary to other meta-heuristics based on local search methods, VNS does not follow a trajectory but explores increasingly distant neighborhood of the current incumbent solution, and jumps from this solution to a new solution if and only if an improvement has been made. Moreover, a local search routine is applied repeatedly to travel from these neighboring solutions to local optima.

In their study, Hansen and Mladenovic (1999) define the basic principles of VNS and state its several applications to five different combinatorial or global optimization problems. They denote with N_k $(k=1,\ldots..,k_{max})$, a finite set of preselected neighborhood structures, and with $N_k(x)$ the set of solutions in the k^{th} neighborhood of x. Their basic VNS begins with the initialization step where neighborhood structures N_k $(k=1,\ldots..,k_{max})$ are selected, initial solution x is evaluated by a local search method or generated randomly, and a stopping condition such as maximum CPU time allowed, maximum number of iterations or maximum number of iterations between two improvements is set. Starting

with the first neighborhood structure, VNS repeats the following steps until the stopping criterion is met.

It generates a candidate point *x'* at random from the k^{th} neighborhood of x and then applies some local search method with *x'* as an initial solution, to obtain a local optima denoted by *x"*. Finally, if this local optima *x"* is better than the current, a move to the point *x"* is performed (x ← x") and continue to the search with the first neighborhood structure. In contrast, if no improvement is achieved at point *x"*, set the neighborhood structure to next one (k ← k+1) and continue.

There are two other types of VNS that differs from the basic VNS namely variable neighborhood descent (VND) method and reduced VNS (RVNS) method. In VND, the change of neighborhood is performed in a deterministic way. Different than the basic VNS, the best neighbor of initial solution x is explored and is denoted as *x'* If this solution *x'* is better than *x*, then *x'* becomes the new x and the search continues with the current neighborhood structure (k ← 1). For the RVNS method, the stochastically generated candidate solution *x'* is directly compared with the incumbent without subjecting to any local search method. If the solution *x'* is superior than incumbent, the search move to the candidate point *x'* (x ← x') and continue with current k structure. If otherwise is true, then the search set (k ← k+1). (Hansen and Mladenovic, 2001)

Although the basic VNS is clearly useful for appropriate solutions of many combinatorial and global optimization problems, it has some inefficiency to solve large instances. Hence, modifications appear to be highly recommended for basic VNS. Two modified version of VNS found in the literature are the Variable Neighborhood Decomposition (VNDS) method that extends the basic form into two level VNS scheme based upon the decomposition of the problem, and the Skewed VNS that addresses the problem of exploring regions of incumbent solutions.

As a change of neighborhood in the search for good solutions to optimization problems is a simple and a very powerful tool. In addition to the extensions of VNS, several authors have incorporated such hybrid features with other meta-heuristics to gain more benefit from VNS. They employ VNS into well-known meta-heuristics such as TS,

GA, SA, GRASP and even constant programming in order to optimize routing, scheduling, traveling salesman, allocation or even clustering problems.

Garcia et al. (2006) propose a heuristic algorithm based on VNS methodology on a linear ordering problem consists of finding a permutation of the columns and rows in order to maximize the sum of the weight in the upper triangle. Their method combines different neighborhoods for an efficient exploration of the search space. For this reason, they construct a hybrid method in which the VNS is coupled with a short term tabu search for improved outcomes.

Lejuene (2006) presents a variable neighborhood decomposition search method for supply chain management planning problem. He employs an algorithm based on VND meta-heuristic which can be considered as a stage-wise exploration of increasingly large neighborhoods. Within each stage, neighborhoods are explored using a branch and bound algorithm.

Hansen et al (2006) developed a VNS heuristic for solving mixed integer programs which provide the origins of VNS method used in hybrid meta-heuristics. They define neighborhoods around the current solution by adding constraints to the original problem, as suggested local branching (LB) method and compare the performance of VNS against LB. Empirical Results show that VNS is simpler and more systematic in exploration and improve 14 times the best known solutions from the set of 29 hard problem instances used to test LB.

Kyöjoki et al. (2005) present an efficient VNS heuristic for the capacitated vehicle routing problem, in which the objective is to design least cost routes for a fleet of identically capacitated vehicles. Their proposed VNS procedure is used to guide a set of standard improvement heuristics and in addition to that a strategy reminiscent of guided local search meta-heuristic is used to help escape from local optimum.

3. TWO TARDINESS BASED SCHEDULING PROBLEMS

Scheduling concerns with the allocation of limited resources to task over time. It is a decision making process that has a goal to optimize one or more objectives. Scheduling problems arise from this optimization effort of limited resources. In a deterministic production planning environment the scheduling problem, is a problem which decides the order of all jobs on each machine and determines the starting time of each job with known ready times and processing times in order to optimize objective function.

More formally, a scheduling problem involves a set of jobs ($j=1,..,n$) and machines ($k=1,..,m$) to process these jobs. Hence, a pair (k,j) refers to processing step or operation of job j on machine k. The following data are associated with job j. (Jain and Meeran, 1999)

- r_j **:** the release time or ready time of the job j. It is the time job j arrives to the shop, that is the earliest time at which job can be processed.
- p_j^k**:** It represents the processing time of job j on machine k. If it is a single machine problem or the processing time for job j is the same on every machine, then the superscript k might be omitted.
- d_j**:** it represents completion time that is promised a customer or an external unit for job j. The completion of a job after its due date is allowed but a penalty is incurred.
- s_{ij}^k**:** this parameter defines the sequence dependent setup time of job j after job i on machine k, meaning that the machine would require s_{ij}^k time unit of setup before processing job j.
- w_j**:** the weight w_j of job j is basically a priority factor, denoting the importance of job j relative to the other jobs.
- S_j**:** it represent the slack time of job j, defined as d_j-p_j-t , where t is the current time.

The scheduling problems are often classified according to their machine number, the scheduling problem with one machine is called single machine problem whereas problems with two or more machines are so called parallel machine or many machine problems. Also scheduling problems can be stated as static or dynamic due to job arrivals.

In static case, a certain number of jobs arrive to the shop simultaneously and shop is ready to start processing immediately. On the other hand, in a dynamic case, the jobs are ready at shop with stochastic or deterministic arrival times. When we are dealing with a deterministic scheduling problem, state of the jobs, due dates, arrival times, processing times and availability of machines are known and do not include any stochastic factor.

In this thesis, we will focus on two hard to solve scheduling problems frequently encountered in practice, namely the Single Machine Total Weighted Tardiness (SMTWT) problem and Parallel Machine Total Tardiness (PMTT) problem, to test the performance of a hybridized Scatter Search algorithm. The detailed description of these two problems together with a brief review of research directed to solving them are given in the following two sections.

3.1. Single Machine Total Weighted Tardiness Problem

Single Machine Total Weighted Tardiness (SMTWT) problem is a static deterministic regular scheduling problem with independent jobs to be sequenced on a single machine with total weighted tardiness measure as the regular optimization criterion. Each job is ready at the shop at time zero. No setup is necessary for the machine before/after processing a job. Each job has a finite processing time, a positive weight and a distinct due date. After generating a sequence for all jobs, earliest completion times C_j and related tardiness values of each job $T_j = max\{0, C_j - d_j\}$ are computed. Each job's tardiness value is then multiplied by its weight and added together to find the sum of the weighted tardiness value. In the literature, the problem is represented as $n/1/\Sigma w_j T_j$ where n denotes the number of jobs, "1" denotes the machine number and last parameter denotes the objective function.

The SMTWT problem is NP-hard (see Lawler, 1977; Lenstra et al., 1977; Du and Leung, 1990) and solution approaches like Dynamic Programming and Branch and Bound are computationally inefficient, especially when the number of jobs is beyond 50, as the results presented in a comparative study by Abdul-Razaq et al. (1990). There is no simple dispatching rule that works best for all problem environments. If there is no more than one tardy job, then the earliest due date (EDD) sequence is optimal, whereas weighted shortest

processing time (WSPT) order gives the optimal sequence when all jobs are necessarily tardy. Therefore, EDD generally performs well for lightly loaded machines while WSPT should be preferred under heavy loading. Several heuristic dispatching rules like those developed by Carroll (1965), Montagne Jr. (1969), Rachamadugu and Morton (1982), Morton et al. (1984), and Panwalkar et al. (1993) have been based on this idea.

The search for good and robust heuristics was continued with sophisticated approaches like meta-heuristics. Matsuo et al. (1989) address the SMTWT by a simulated annealing algorithm which starts with a good initial solution and low acceptance probability to accelerate the search for a near optimal solution. Potts and Van Wassenhove (1991) propose a descent heuristic and a simulated annealing method for SMTWT. Crauwels et al. (1998) present single and multi-start versions of descent, simulated annealing, Tabu Search (TS) and genetic algorithm implementations for the same problem and show that while simulated annealing is outperformed, Tabu Search dominates the other methods. Congram et al. (2002) treat SMTWT with an 'iterated dynasearch' algorithm, which is a local search technique that uses dynamic programming to find the best move which is composed of a set of independent interchange moves and searches an exponential size neighborhood in polynomial time. They obtain results that are superior to other local search procedures. Laguna et al. (1991) consider a single machine scheduling problem for minimizing the sum of setup costs and linear delay penalties, and propose a TS algorithm that uses hybrid neighborhood consisting of both swap and insertion moves.

In his study, J. Schaller (2004) presents a timetabling algorithm that inserts idle time into a given job sequence on a single machine in order to minimize the sum of the absolute value of the lateness at jobs to be scheduled. The single machine problem investigated composed of two set of problems; one is the single machine scheduling earliness/tardiness problem and the other is a problem involving a quadratic measure of performance optimization for single machine scheduling. Timetabling algorithm is used on partial sequence in Branch and Bound search. It is also modified so a lower bound on objective value due to the jobs that have not been dispatched can be obtained for a partial schedule. Later, three different B&B procedures are performed to minimize the objective and their results are compared to draw a conclusion.

Azizoğlu et al. (2003) present a heuristic that minimizing maximum earliness while keeping the number of tardy jobs to its minimum value for an earliness/tardiness single machine problem. They propose a general procedure to generate all efficient schedules for biciriteria problems and then develop a method to find the best schedule that minimizes a composite function of two criteria problem by evaluating only a small fraction of previously generated efficient solution set.

In their study, Feldman and Bishop (2003) consider a problem of scheduling a number of jobs on a single machine against a restricted common due date. According to complexity of restricted common due date problem, it is unlikely to find an efficient integer programming algorithm. Hence, Feldman and Bishop develop a new and appropriate problem representation and apply three different meta-heuristics namely evolutionary algorithm, simulated annealing, and threshold accepting. They demonstrate the efficiency of meta-heuristics against integer programming by obtaining near-optimal solutions.

Kethley and Alidaee (2002) examine various scheduling rules, heuristics and algorithms including WSPT rule, variation of the modified due date rule, a genetic algorithm, neighborhood job search for the problem of scheduling n jobs in a single machine to minimize the total weighted late work (TWLW). In their search, Kethley and Alidaee mostly concentrate on two hypotheses evaluated during the simulation of predefined search procedures. According to first hypothesis, there is no difference in the Total Weighted Late Work generated by WSPT, combination rule (CBN), neighborhood job swap (NJS), and GA. For the second hypothesis, there is no difference in TWLW generated by the various initial due date and deadline parameters. They conclude that the performances are same regardless of parameter settings for due date and deadline.

In his study, Franca et al. (2001) introduce a new memetic algorithm (MA) for the total tardiness single machine scheduling problem with due dates and sequence dependent setup times. The main contributions with respect to the implementation of the hybrid population approach are a hierarchically structured population concerned as a tenery tree and the evaluations of three recombination operators. Concerning local improvement procedure, several neighborhood reduction schemes are developed and proved to be

effective when compared to the complete neighborhood. They judge pure GA and MA against a multi-start algorithm that employs the all pairs neighborhood as well as two constructive heuristics.

Bilge et al. (2007) present a TS approach to the single machine total weighted tardiness problem. The problem investigated in the study, consists of a set of independent jobs with distinct processing times, weights and due dates to be scheduled on a single machine. While minimizing total weighted tardiness, a totally deterministic TS algorithm with a hybrid neighborhood and dynamic tenure structure is employed. In addition to that, the strength of the several candidate list strategies is considered in order to increase the efficiency of the search. The proposed TS approach yields very high quality results for a set of benchmark problems obtained from literature (Crauwels et al., 1998).

3.2. Parallel Machine Total Tardiness Problem

The classical parallel machine total tardiness problem (PMTT) can be defined as the scheduling of *n* jobs on *m* continuously available identical parallel machines aiming to minimize total tardiness. Each job is processed on an assigned machine as long as its processing time. Each machine can process only one job at a time, and each job can be processed on only one machine. Each job is ready at the beginning of dispatching process and has a distinct processing time and a due date. In most of the PMTT problems, machine available on job-shop are identical, jobs are ready at time zero and setup times for consecutive jobs are ignored. After assigning all the jobs on available machines, total tardiness is evaluated where tardiness is the amount of time that completion time exceeds due date.

PMTT is NP-hard even for a single machine (Du and Leung, 1990) and exact methods are mostly limited to special cases like common due dates and equal processing times (i.e. Root 1965, Lawler 1977, Elmaghraby and Park 1974, Dessouky 1998). Recently, Liaw et al. (2003) present a Branch and Bound algorithm that incorporates various dominance rules along with efficient lower and upper bounds for the case of unrelated machines, distinct due dates, zero ready times and no setup times, and report that the algorithm performs well up to 18 jobs and four machines.

In this study, we deal with a generalized PMTT problem that introduce complex real world situations such as; distinct ready times, uniform parallel machines with different processing speeds and sequence dependent setup times. In this generalized PMTT, there are n jobs to be processed on m machines of k types. Machines belonging to same type are identical where machines belonging to different types are uniform. Each job j has an integer processing time p_j^k on type k machine, an integer ready time r_j, a distinct due date d_j, and a sequence dependent setup time s_{ij}^k of processing job j after job i on a type k machine. For a given sequence of jobs, the earliest completion time and related tardiness is computed for each job: $T_j = \max \{0 , C_j - d_j \}$. Hence the aim is to find a dispatching order that minimizes the total tardiness. Although meta-heuristic approaches to scheduling problems in general are quite abundant, the literature is sparse for our specific generalized version of PMTT.

Ergun et al. (2002) present a new local search heuristic based on combining variable number of insertion moves for the parallel machine total weighted completion time scheduling problem. In their study, they introduce a very large scale neighborhood search that applies a set of insertion moves for identical parallel machines. Also using the special structure of the scheduling problem, they develop a very efficient variable depth search method based on multi-label keeping shortest path algorithm. In their computational study, they compare the performance new heuristic with the various search frameworks including steepest descent, multi-start TS, and iterated local search. According to their findings, the new variable depth sequential insertion neighborhood heuristic with the iterated local search procedure is the most effective among all.

Sivrikaya-Şerifoğlu and Ulusoy (1999) employ two GA approaches for parallel machine scheduling problem with earliness/tardiness penalties. One of them is a GA with new cross-over operator which is developed to solve multi-component combinatorial optimization problems and the other is a GA with no cross-over operator. Based on result of 960 randomly generated tests, they conclude that neighborhood exchange accomplished by the mutation of GA can yield relatively better results in a small and easy instances of problems whereas GA with new cross-over operator outperforms when large sized, more challenging problems occur.

Bilge et al. (2004) propose a totally deterministic TS approach to PMTT problem introduced by Sivrikaya-Şerifoğlu (1999). Because of the complex nature of the problem, they employ a hybrid neighborhood generation method, with several candidate list strategies and intensification/diversification phases. In their study, they come up with a "low" candidate list strategy that considers job insertions from the machine with maximum contribution to total tardiness to each of the other machines, thus isolating desirable regions of the neighborhood and increase the speed of the search. Their TS algorithm performs much superior compared to previously cited results.

In their research paper, Bilge and Kıraç (2006) presents an adaptive GA for PMTT problem. The adaptive control mechanism mainly focuses on slowing down or preventing the premature convergence and reduces the parameter dependence of basic GA. Population diversity is selected as the key factor for the search and close-loop controllers are employed to achieve the desirable population diversity and quality when it decreases below a certain threshold. These controllers apply a series of insertion moves in the form of mutation and smooth out the peakedness of population distribution while pushing the search into different regions.

Anghinolfi and Paolucci (2006) propose a hybrid meta-heuristic approach which integrates several features from TS, SA, and VNS in a configurable scheduling algorithm for the PMTT problem. They combine the idea of SA which states that randomness in generating candidate solutions could improve the search effectiveness, the principle of TS that an intelligent neighborhood exploration must be guided by a appropriate candidate list strategy, and the concept of VNS that changing the neighborhood structure during the search might avoid to get trapped at premature local optimum. The HMH method outperforms previous methods employed for PMTT problems introduced by Sivrikaya-Şerifoğlu (1999).

4. THE PROPOSED HYBRID CONTINUOUS SCATTER SEARCH APPROACH

Our Hybrid Continuous Scatter Search (HCSS) approach is a hybrid meta-heuristic method which integrates Scatter Search and Variable Neighborhood Search in a new configurable scheduling algorithm. Scatter Search used in our algorithm, serves as a diversification mechanism which ensures that different promising regions of solution space is visited, whereas variable neighborhood search method exploits systematically the idea of intensification in descent to local minima. These two methods are employed interchangeably depending on the performance of the search. The detailed description of HCSS approach for the SMTWT and PMTT problems considered in this thesis is provided in the following sections.

4.1 HCSS Approach to SMTWT

Single Machine Total Weighted Tardiness problem is a static deterministic regular scheduling problem with independent jobs to be sequenced on a single machine with total weighted tardiness measure as the regular optimization criterion. Each job is ready at the shop at time zero. No set up is necessary for the machine before/after processing a job. Each job has a finite processing time p_j, a positive weight and a distinct due date d_j. After generating a sequence for all jobs, earliest completion times C_j and related tardiness values T_j of each job j are evaluated using the formula $T_j = \max\{0, C_j - d_j\}$. Each job tardiness value is then multiplied by its weight and added together to find the sum of the weighted tardiness value. In accordance with the definition of SMTWT problem, our HCSS approach is introduced in the following sections.

4.1.1. Solution Encoding Scheme

The most important issue when applying Scatter Search successfully to SMTWT problem is to develop an effective problem mapping and a solution generation mechanism. If these two mechanisms cooperate efficiently, it is possible to find a good solution for a given optimization problem in an acceptable time. Our HCSS approach is a continuous

search method that directly handles vectors of real component and combines these vectors by linear combinations to produce new ones through successive generations. It aims to map each and every solution hidden in the search space with proper vector representations and to find good solutions by using search procedures based on basic vector operations.

In order to construct a suitable mapping between problem solution and SS vectors, start time of each job is used as the real number component of the vectors. Each vector consists of $n+1$ entities where n denotes the number of jobs to be scheduled on a single machine. These n jobs have distinct process times, weights and due dates. Therefore, every job is denoted with a unique job index and corresponding start time. In a solution vector, first n positions contain the start times of n jobs, such that the i^{th} position in the vector corresponds to the start time of job with index i. The last $n+1^{st}$ position of the solution vector is dedicated to total weighted tardiness value (*twt*) related to these start times. Figure 4.1 depicts an example, where *job # 1* starts at time 1428, *job # 2* starts at 1843 and finally *job # N* starts at 1881 and *twt* value is 330.

JOB # 1	***JOB # 2***	***.....***	***JOB # J-1***	***JOB # J***	***JOB # J+1***	***.....***	***JOB # N-1***	***JOB # N***	***FIT.***
1428	1843	***.....***	1610	162	901	***.....***	818	1881	330

Figure 4.1. Solution encoding for single machine total weighted tardiness problem

In this vector representation, jobs are not sorted in an increasing order of start times that means job # 1 would not be processed in the first place at the specified start time. Any job with a smaller start time has the priority to be processed earlier than the other jobs that have larger start time values. The *twt* value is calculated by scheduling n jobs according to a permutation sequence obtained after sorting jobs with respect to non-decreasing start times. If a vector corresponds to more than one permutation, the one yielding minimum *twt* value is selected. Thus, each solution vector represents a unique sequence with a *twt* value that defines its position in the solution space. Conversely, each job permutation with unique *twt* corresponds to many solution vectors in the continuous search space. Consider a solution where three consecutive jobs' start times are 97, 228 and 322, respectively. The start time of the job in the middle can have a value between 97 and 322 without altering the corresponding permutation. This also means that a given permutation can be reached from many different points in the continuous search space and allows an interesting flexibility

property in terms of neighborhoods defined on the solution space. By using linear combinations of start times, we can express complex neighborhoods of a considered sequence in the search space.

Another advantage of the start time encoding appears when dealing with the basic vector operations such as addition, subtraction or even multiplication with constant parameters. Easy implementation of these basic operations will help us in defining a solution combination method which is used to generate new trial solutions and distance measurement methods to select diverse solutions for reference set. These two methods will be discussed in further sections.

4.1.2. Initial Solution Generation Method

One of the most crucial component of our HCSS approach is the initial solution generation method. SS method is based on generating new solutions by creating numerically weighted combinations of existing solutions. Therefore, the initial population which is carrying the ancestors of the next generations plays a leading role. The future can not be predicted by looking at the past. In other words, poor solutions sometimes may lead us to better solutions, whereas an initial population containing only good solutions does not always guarantee a better result. Hence, the most essential feature of a proper initial solution generation method is to create a collection of both diverse and good solutions.

In our HCSS approach to SMTWT problem, three different initial solution generation methods are employed and compared in order to acquire the most efficient initial population. The initial population contains predefined number of individuals. The first approach uses a simple generator that orders jobs randomly and then evaluates their *twt* values by scheduling them at their earliest start times.

Second approach employs simple heuristics to create some seeded solutions and a random sequence generator to fill the remaining pool with diverse trial solutions. Heuristics utilized in this approach are EDD, SPT, WSPT and R&M rules. Before going into details of the second approach, it will be a good idea to provide brief overviews of the

mentioned heuristics. The following terminology helps to gain an understanding about their nature and functions.

- *EDD* (Earliest Due Date) is a dispatching rule that arranges jobs in increasing order of due dates.
- *SPT* (Shortest Processing Time) is a dispatching rule that arranges jobs in increasing order of processing times.
- *WSPT* (Weighted Shortest Processing Time) is the weighted version of SPT.
- *R&M* (Rachamadugu and Morton, 1982) is based on sorting the jobs in order of non-increasing priorities that are evaluated dynamically by predefined formulas.

WSPT heuristic considered in this study is first introduced by Montagne (1969), which use basic full priority WSPT multiplied by a slack factor. The slack factor is close to 1.0 for very early due date jobs and very close to zero for very late due date jobs. Slack factor employed here is not dynamic and even if a job is overdue, it is still not given a full WSPT priority. The priority formula is as follows;

$$\pi_j = (w_j / p_j)[1.0 - (d_j / \Sigma_i p_i)] \tag{4.1}$$

where π_j is the priority of job j, w_j is the weight, p_j is the processing time and d_j denotes the due date of job j. The jobs are sequenced according to their non-increasing order of priorities and their related *twt* values are evaluated.

R&M heuristic for total weighted tardiness problem, developed by Rachamadugu and Morton (1982), is based on sorting the jobs in order of non-increasing priorities. The priority π_j is obtained by using the formula;

$$\pi_j = (w_j / p_j)[\exp\{-(S_j)^+ / kp_{av}\}] \tag{4.2}$$

where k is a factor and p_{av} is the average processing time of all jobs to be scheduled. The slack time S_j^+ at time t can be computed as follows;

$$S_j^+ = \max(0, d_j - p_j - t) \tag{4.3}$$

As time t indicates the current time, this heuristic uses a dynamic procedure where priorities π_j must be updated after a job with the highest priority is scheduled and its completion time on machine is computed. When all the jobs are sorted accordingly, the final total weighted tardiness value is evaluated. In this study, a small modification is applied to R&M heuristic and k factor becomes no longer a fixed value. The solutions are generated using each of the k values in the range [0.5, 4.0] with increments of 0.1.

Our second population generation method seeds all solutions created by EDD, SPT, WSPT, and modified R&M heuristic into the initial population the rest of which is filled using a random generator as in the first approach. Third method is a variant of the second method in which 10 best R&M solutions together with EDD, SPT and WSPT solutions are established in the initial pool. In all three methods duplication is avoided by replacing solutions that have exactly the same set of start times. Another screening mechanism tried during numerical experimentation, eliminates solutions with same *twt* values even if their start times are different.

These three initial solution generation methods will be compared via numerical experimentation and a final choice will be made according to their contribution to the ultimate outcome.

4.1.3. Reference Set Update Method

The reference set update method plays the key role in HCSS approach. As previously defined, the reference set is used to produce new solutions by applying linear weighted combinations and it is a collection of both high quality elite solutions and structurally diverse solutions selected from a pool. In the first iteration, the pool contains initial solutions generated using one of the three approaches described in the previous section. In the subsequent iterations, the pool consists of solutions created by a combination method. The number of elite solutions b_1, and diverse solutions b_2 in the reference set are fixed and determine its size ($b=b_1+b_2$). Two methods are devised to select the b_1 elite solutions.

In our first method, the best solutions selected from the pool are stored in a list called the reserve list. This list behaves as a long term memory of HCSS by keeping best known solutions visited so far. At the end of each iteration, solutions kept in the reserve list are updated if any superior ones are generated. Therefore, reserve list holds only of b_1 elite solutions which are defined as good generators. While revising the reserve list, we keep track of entering solutions in order to make sure that new entries are not identical twins of existing solutions. If at least one start time of any job *j* in the entire vector for the candidate differs from the corresponding start time of the same indexed job contained in other existing solutions, then candidate is permitted to replace a poorer solution from the list. After the authorized substitutions, the b_1 of best solutions are transferred from the reserve list to the reference set. This method leads to a 2-tier design for the reference set, i.e. elites from the reserve list and diverse solutions from the pool.

Create initial solution set

schedule jobs with respect to EDD and store the trial solution x_{EDD}

schedule jobs with respect to SPT and store the trial solution x_{SPT}

schedule jobs with respect to WSPT and store the trial solution x_{WSPT}

for k = 0.5 : +0.1 :4.0
schedule jobs with respect to R&M and store the trial solution x_k if fitness $x_k \neq$ fitness { x_{EDD}, x_{SPT}, x_{WSPT}, x_{k-1} }
end

select either all or best 10 x_k

create remaining trial solutions using a generator that schedule jobs randomly x_{RAND}

delete one of the two trial solution having the same fitness value

the initial solution set = {x_{EDD}, x_{SPT}, x_{WSPT}, x_k, x_{RAND}}

Figure 4.2. The pseudo-code of initial solution generation method for the third approach

In the second method, the updating procedure of the elite set employs a 3-tier design (Laguna and Marti, 2006), where the first tier consists of high quality solutions obtained from the pool, the second tier consists of best solutions copied from the reserve list and finally the third tier includes the diverse solution selected by a distance

measurement method. The second update method is aiming to preserve diversity continuously, instead of allowing it to become homogenous by only admitting solutions from one source, which tend to have very similar components at further stages of the search. In the first method, reference set best solutions are rooted from the reserve list. In other words, b_1 solutions stored at reserve list are directly copied to reference set without considering their structures or past performances. These individuals may be neighbors of the same local optima or may remain in the list for a long time. In such a case, it directly affects the reference set best solution variety and indirectly influences the offspring generated from these elite pairs.

Update Reserve List

POOL = $(x_1,.....,x_k)$, let x_i be the trial solution and $x_i =< st_1,….st_n>$ where st_j is the start time of job j.

ReserveList = $(y_1,.....y_b)$ where y_1 is the best and y_b is the worst solution stored in list.

$y_1 = < st_1,….st_n>$

```
select xi's such that fit.(xi) <= fit (yb)                    // fit.= fitness of a sol'n
    for all selected xi's
        if fit.(xi) <= fit (yb)  /
            for all yi                                          // (i = 1,….,b)
                if any stj of xi ≠ any stj of yi                // (j = 1,….,n)
                if fit.(xi) ≠ fit (yi)   // used in screening method
                    yb = xi
                    sort ReserveList in an increasing order of fit.
                    update yb
                    delete xi from POOL
                end
            end
        end
    end
```

Figure 4.3. The pseudo-code for reserve list update method

Before starting the selection step of second method, reserve list is updated according to the rules used in the first case, and the solution pool is rearranged in order to find high quality solutions. Different from the first method, half of the b_1 solution is transferred from the reserve list and the other half is chosen from the high quality members

of the pool. In this way, we assure at least some of the best solutions to be dissimilar than the ones employed in the previous iteration. Additionally, same screening mechanism used in initial solution generation method is again utilized to increase the diversity of both the reference set and reserve list. Hence, two solutions with the same *twt* value can not enter the elite set at the same time.

After deciding the b_1 elite solutions, the next step is to select b_2 diverse solutions. For each solution left in the pool, its distance to all members of the current reference set (initially, only b_1 elite solutions) are calculated. The minimum of these distances gives the distance of the candidate to the reference set. The candidate with maximum distance to current set is selected as a diverse solution and added to the set. The chosen solution is deleted from the pool and preceding minimum distances to the set are updated with respect to newly added diverse solution. The process is repeated until all b_2 diverse elements are determined. Two different types of distance measurement methods are devised; these are *start distance* and *rank distance* measurement methods. Based on the distance measure employed, three diverse selection procedures are introduced; start diverse, rank diverse and mix diverse selection procedures. In mix diverse selection procedure, start distance method is initially executed and half of b_2 diverse solutions are selected and deleted from the pool. Then the rank distance measure is applied for the remaining solution left in the pool and the other half of b_2 is determined accordingly. These three procedures will be compared to decide the most effective diverse element selection technique. We describe the distance measurement methods that were developed within the context of this thesis below:

Start distance measurement method is inspired from a well known rectilinear distance formula used in real space vector problems. In our solution encoding, start times correspond to the coordinates of a position vector that expresses a point in the real space, and our start distance measure is given as

$$\text{distance} = |(st_{11}-st_{21})| + |(st_{12}-st_{22})|+\ldots+|(st_{1j}-st_{2j})|+\ldots\ldots\ldots+|(st_{1n}-st_{2n})| \qquad (4.4)$$

where st_{ij} denotes the start time of a job j for the solution vector i. Although the total distance measured by this modified version indicates a relative value instead of a real one,

it is sufficient for the selection procedure to work properly. The details of start distance measurement method are described in Appendix A.

The rank distance measure originates from a puzzle where a shuffled string of numbers should be rearranged with minimum restricted insertion moves to catch a winning sequence. For an accurate adaptation, a chain of numbers is required rather than the values, but our vector representation contains only the start times and the total weighted tardiness value. Hence, start times are sorted in non-decreasing rank and their job indices are noted in the same order. The solution taken from the pool is denoted as shuffled string and the reference set solution respect to which the distance is measured, cited as the wining sequence. The restricted insertion move indicates that a job can only be inserted between its two preceding jobs. The other insertion moves are not allowed. A simple algorithm is employed for rank distance measurement method that computes minimum moves needed to convert a shuffled string into the winning sequence. Finally, the number of total performed moves states the rank distance between a pool solution and a reference set solution. The details of rank distance measurement method are described in Appendix B.

4.1.4. Subset Generation Method

The simplest form of subset generation method that is used in our approach consists of generating all pairs of reference set solutions. The method concentrates on subsets of size two resulting in $(b^2-b)/2$ newly generated subsets where $b=b_1+b_2$. The pairs in new subsets are selected one at a time in lexicographical order and solution combination method is applied to produce a trial solution.

4.1.5. Solution Combination Method

Solution combination method also known as combination mechanism or crossover is a common approach used in most evolutionary algorithms to create new solutions. The solution combination mechanism is a method for sharing information between solutions. Generally it combines features of two parents to form several offspring with the possibility that good solutions may reproduce superior ones. In scatter search, the importance of the combination method is even greater due to its strong impact on the exploration power.

Since population diversity is obtained by creating new solutions, it induces reliability in the search process.

Update Reference Set

$POOL^{*} = (x_1,.....,x_k)$, where * denotes that this set is updated after deleting x_i used in reserve list update step

$ReserveList^{*} = (y_1,.....y_b)$ where * denotes that this set is updated with new best solutions entered from the POOL

$RefSet = (best_1,...., best_b, div_1,....,div_b)$ where $best_i$ represent best solution and div_i is the diverse solution selected to RefSet.

RefSet = $ReserveList^{*}$ // initial design

RefSet ($best_1,...., best_{b/2}$) = $ReserveList^{*}$ ($y_1,.....y_{b/2}$) // 3-tier design

RefSet ($best_{b+1},...., best_b$) = best b/2 solution of $POOL^{*}$ // 3-tier design

while # of selected div_i < = b

for all x_i € $POOL^{*}$

for all Refset sol'n

DistMatrix(x_i, RefSet sol'n) = start distance between x_i and RefSet sol'n **

DistMatrix(x_i, RefSet sol'n) = rank distance between x_i and RefSet sol'n **

update row minimums for DistMatrix

find x_{max} with max of row minimums

RefSet (div_i) = x_{max}

delete x_{max} from $POOL^{*}$

end

end

** executed interchangeably

Figure 4.4. The pseudo code for reference set update method

In our HCSS approach, we use BLX-α operator which is one of the most effective combination methods developed for real-coded continuous GAs. Herrera et al. (2006) carry out empirical study of different combination method instances for real coded evolutionary algorithms and their experiences on the application of solution combination methods show that the BLX-α operator outperforms the other methods. They have considered BLX-α as a combination method due to three facts: it includes randomness which can be effective, it favors the production of diversity in the population of an EA which may improve the reliability while avoiding premature convergence and finally, BLX-α has a self adaptive

behavior that can generate offspring according to the distributions of parents without any control parameter.

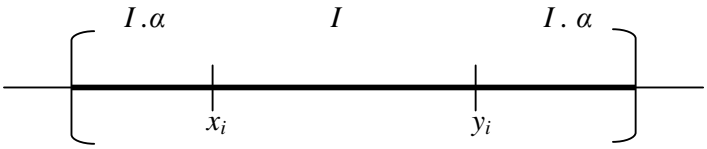

Figure 4.5. BLX- α combination method

Let us assume that $X=(x_1,\ldots, x_n)$ and $Y=(y_1,\ldots, y_n)$ $(x_i, y_i \in R, i=1,\ldots,n)$ are two real-coded vectors selected to be combined. BLX-α combination method generates an offspring $Z=(z_1,\ldots, z_n)$ where z_i is a randomly chosen number of the interval $[c_{min} - I.\alpha, c_{max} + I.\alpha]$ where $c_{max}=max(x_i, y_i)$, $c_{min}=min(x_i, y_i)$ and $I= c_{max} - c_{min}$ and α is a constant value. Revisiting our solution encoding, any pair of parent solution can easily be defined as X and Y vectors where x_i and y_i represents the start times for each job. Also, the Z vector becomes the new trial solution with z_i's denoting the imaginary start times. An illustration of solution combination method is given in Appendix C.

Investigations demonstrate that BLX- α with α=0.5 performs better than other BLX- α operators with any other α value. Herrera et al. (2006) report that BLX-0.5 offers a useful tool to enhance global search (exploration) capabilities of continuous SS and induce reliability in the search process. As previously mentioned, our approach is a continuous SS which directly handles vector of real components and combines these vectors by linear combinations to create new ones trough successive generations. Therefore, BLX-α operator is a perfect match for our solution combination method. Additionally in further stages, we will test dynamically changing α value instead of fixed α attempting to regulate diversity of newborn individuals.

Offspring created by solution combination method are stored in a transition set called pre-pool. Pre-pool contains trial solutions with temporary start times that have no total tardiness value. Hence, a solution improvement method is applied to the pre-pool solution. This method sorts the jobs with respect to non-decreasing start times and calculates the fitness value without changing the temporary start times. A screening

mechanism is employed and one of two offspring with same *twt* value is deleted instantly from the pre-pool. Finally, filtered trial solutions are arranged in increasing order of their fitness values and transferred into the main pool. As previously indicated, pool provides a source for reference set update method. Best and diverse solutions are both selected from the pool. Therefore, its condition highly affects the structure of the updated reference set and future offspring. Poor diverse solutions selected continuously from the pool may lead the reference set far from the desired optimum, and reduced its skill to produce qualified next generations. On the other hand, a pool lacking diverse solutions may cause a premature convergence. In absence of diverse individuals, the selected individuals are more or less the same as their elite counterparts acquired from the same source. So the offspring generated from these similar parents, most probably result in identical children, and the search will be trapped at a local optimum without visiting all of the promising regions potentially contain optimum solutions. In order to overcome these handicaps, we introduce the parameter pool size that determines the number of trail solutions accepted to the pool. Three different strategies are investigated for the pool size; an enlarged size pool having every solution generated including the very poor diverse ones, a default size pool containing best 100 fitness value trial solutions and a reduced size pool holding best 85 trial solution excluding most of the poor members. These strategies are examined in the numerical experimentation section and the best one is selected for our HCSS approach.

4.1.6. Variable Neighborhood Search as an Intensification Strategy

The basic concept of neighborhood search method is quite simple. One starts with a feasible solution to a problem and the solutions within a neighborhood of the current solution are evaluated. If one of these solutions is better than the current solution, it becomes the new current solution and its neighborhood is investigated until no improvement can be found. The current solution obtained finally is the local optimum. Therefore, neighborhood search is a very focused and has been referred to as exploitation or intensification method. Different from general neighborhood search methods, VNS visits several neighborhoods instead of a single one, where typically neighborhoods move further and further away from the current solution depending on the search depth. In our HCSS approach, VNS is employed as an intensification strategy that examines the neighborhood of elite solutions.

Solution Combination Method

Let (f , m) be a pair of solutions and (f_i , m_i) Є SubSet where SubSet is a set containing all pairs of RefSet generated by subset generation method. And offs=<st_1^c,….st_n^c> is the offspring solution.
f = <st_1^f,….st_n^f>, st_j^f is the start time of job j for solution f.
m=<st_1^m,….st_n^m>, st_j^m is the start time of job j for solution m.

```
for all (f , m) Є SubSet                          // total (b^2-b)/2 pairs where b is the size of RefSet
    for all j                                     // j = 1,.....,n  n denotes the # of jobs
        p_min = min(st_j^f, st_j^m)
        p_max = max(st_j^f, st_j^m)
        interval = | st_j^f - st_j^m |
        c_min=p_min-(interval*α)                  // α is constant
        c_max=p_max+(interval*α)
        offs (st_1^c) =c_min+(rand*(c_max-c_min))  // rand is a random number between 0.0 & 1.0
    end
    prePOOL(i) = offs                             // i = 1,....., (b^2-b)/2
end
for all trial sol'n Є prePOOL
     sort  jobs in order of non-decreasing start times
     obtain sequence of jobs
     schedule jobs and calculate fitness
     copy trial sol'n with its fitness to POOL
end
delete one of the two solution from POOL with the same fitness
if POOLCap > size(POOL)                           // POOLCap is the pool size
    POOL =POOL
else
     POOL=POOL (1:POOLCap)                        // most fit  trial  soln's are selected
end
```

Figure 4.6. The pseudo code for solution combination method

The neighborhoods are simply generated by using swap moves. A swap move exchanges the location of two jobs so that each job is placed in the position previously occupied by the other. In our real coded solution representation, the start times determine the sequence of jobs. So having a sequence, SMTWT can be considered as a permutation problem. In order to create a neighborhood, VNS divides the sequence into several subsequences where each subsequence has a number of jobs determined by the search depth. Starting with the first group including the jobs that will be processed earlier, VNS lists all possible permutations of these jobs. The permutations are found by using swap

moves. The initial order of jobs within the selected group is replaced with one of its alternative permutation arrays without changing the remaining sequence. Evaluating all alternative permutations, the total weighted tardiness of each newly generated sequence is computed. If one of these performs better than the initial solution sequence then the initial sequence is revised and VNS moves to the next subsequence.

After all the subsequences are investigated in the same way, if the final sequence obtained is an improved one, it becomes the new current solution and the search depth contracts in size, i.e. the subsequence size is reduced by one. Otherwise, the current solution is kept and the search depth expands in size. Expansion or contraction of the search depth means a change in the definition of the neighborhood. At the final stage, VNS examines the elite solutions' neighborhoods with the largest search depth and depending on the outcome it either stops or continues to search. If it can not improve the current fitness value, then the intensification procedure is terminated and the current solution becomes the local optimum.

Six different VNS approaches are developed to attack the SMTWT problem. The first issue is related to the neighborhood definition, or the search depth used in subsequence sizing. In our three step search depth VNS approach, VNS 3-4-5, and VNS 4-5-6 are employed to seek neighborhoods of treated solutions. VNS 3-4-5 denotes that the VNS starts to divide the sequence into subsets of three consecutive jobs to generate alternative solutions and the size is increased to four if no improvement is achieved with the previous search depth. Final run is performed using subset of five jobs if no achievement is experienced so far. Similar to VNS 3-4-5, VNS 4-5-6 behaves in a same way. The only difference is the size of each consecutive search depth utilized in VNS.

The second issue is related to the timing of VNS method and two approaches, Final VNS (FVNS) and Middle VNS (MVNS), are developed to investigate the best timing for a neighborhood search. Final VNS as its name suggests, is executed at the end of SS algorithm. Usually, SS completes its search throughout the solution space and comes up with a best solution. In this method, the ultimate elite solution is attacked with FVNS and its neighborhoods are visited in order to obtain an improvement. On the other hand, MVNS is implemented after a number of iterations defined by a counter. This counter is so called

best solution VNS counter. It starts with the first iteration and counts successive iterations passed without any improvement of the best known. Best solution VNS counter zeros itself if SS or VNS finds a better solution for the problem. After exceeding a certain threshold value the counter triggers VNS switch and middle VNS is executed. At the end of MVNS, our counter is either set to zero or continues to count depending on the achievement of an improvement.

Another issue is related to the selection of elite solutions to which MVNS is applied. MVNS is applied either to best three or to all elite solutions of the reference set. Employing VNS for all best solutions seems a time consuming operation compared to the application of its restricted version. In contrast, without investigating all elite solutions, better outcomes hidden at the neighborhood of these solutions may be missed. Therefore, the critical discussion, time vs. performance arises, and will be discussed in experimental section. The details of VNS method is described in Appendix D.

4.1.7. Alpha Strategies

Nomura et al. (2001) have demonstrated theoretically that BLX-α has the ability to promote diversity in the population of an EA. They state that BLX-α spread the distribution of chromosomes when $\alpha > (\sqrt{3}-1)/2$, reducing it otherwise. Nomura et al. (2001) observe BLX-0.0 makes the variances of the distribution of the chromosomes decrease, reducing the distribution whereas BLX-0.5 causes the variances of distributions increase while spreading them. In this way, BLX-α provide useful tool to enhance exploration capabilities of continuous SS. In our approach, we introduce two strategies; a static α value of 0.5 and dynamically changing α values, in order to control diversification mechanism. In the first strategy, α is set to 0.5 which is proven as the best among other constant values. In the second one, three different α sub-strategies are developed to test the effects of dynamic α on solution combination method.

In the first dynamic strategy, α value follows a loop where it starts with 2.0, then drops to 0.5 after certain iteration, increases incrementally at the end of non-improved iterations and finally turns back to its initial value. The whole changeovers are controlled by a switch. Alpha switch is a binary parameter which becomes 1 (on) if VNS does not

improve the best-known solution or becomes 0 (off) if otherwise is true. At the end of each iteration, the condition of alpha switch and α value are updated depending on the performance of VNS or SS. An improvement makes the switch off and set α value back to 2.0. For the counter case when the alpha switch becomes on; α value drops to 0.5 and during the further iterations where the condition of the switch is unchanged, it is increased by 0.005 at the end of each iteration unless a better global optimum is found.

In the second dynamic strategy, α is fixed at 0.5 during the period where alpha switch is off. When it becomes on, α is increased by 0.005 at the end of every iteration without any improvement of the best known solution. The value of α returns to its initial value 0.5 if SS or VNS improve the best known.

In the last dynamic strategy, α is again fixed at 2.0 and drops to 0.5 when the alpha switch turns into on. But this time, it stays fixed at 0.5 at the end of every iteration while the alpha switch is still on. The value increased back to 2.0 if an improvement is succeeded. The new hypothesis behind developing dynamic α values is to control diversity and broaden the search to different promising zones when the reference set solutions start to generate less diverse offspring. This hypothesis will be tested in experimental results section.

4.1.8. Stopping Criterion

Two stopping criteria are investigated for our HCSS algorithm; a fixed iteration criterion and a dynamic stopping criterion. According to the first condition, the algorithm stops at the end of a predefined iteration number whereas for the dynamic stopping criterion, best solution VNS counter is again employed and the algorithm continues to operate until the counter of non-improving iterations reaches a certain value. The best value of the fixed iteration number and the threshold value for best solution VNS counter are determined through preliminary experiments. The experiments related to fix versus dynamic stopping criteria are discussed in further sections.

Variable Neighborhood search

Let X be an elite solution of RefSet such that X = <st_1, st_2,….., st_n> where st_j is the start time of job j. and globalBest is the fitness of current solution. Suppose that VNS 3-4-5 is applied to X where 3-4-5 indicates the three different search depths.

```
vns_depth = 3
vns_switch=1
sort X in order of increasing start times and obtain the sequence S      // S is an array of job #'s
sorted currentSol = S
globalBest = fitness(S)
while vns_switch=1
        if vns_depth=3 (or 4 or 5)
                divide jobs into groups where each group has 3 (or 4 or 5) jobs
                for each groups
                        write all permutations of jobs within the group
                        for each permutation
                                generate neighborhood S_i*
                                evaluate fitness (S_i*)
                        end
                        select S_best* with smallest fitness value
                        if fitness(S_best*) < fitness(S)
                                S = S_best*
                        end
                end
        if vns_depth=3
                if fitness(S) < globalBest
                        globalBest= fitness(S)
                        sorted currentSol = S
                        vns_depth=4;
                        vns_switch=1;
                else
                        vns_depth=4;
                        vns_switch=1;
                end
        elseif vns_depth=4
                if fitness(S) < globalBest
                        globalBest= fitness(S)
                        sorted currentSol = S
                        vns_depth=3;
                        vns_switch=1;
                else
                        vns_depth=5;
                        vns_switch=1;
                end
        else vns_depth=5
                if fitness(S) < globalBest
                        globalBest= fitness(S)
                        sorted currentSol = S
                        vns_depth=3;
                        vns_switch=1;
                else
                        vns_switch=0;
                end
        end
end
evaluate start times and fitness value of new solution X* using sorted currentSol.
```

Figure 4.7. The pseudo code for variable neighborhood search method

```
create initial solution set P
POOL = P
RefSet (best1,…., best_b1 ) = best b1 solution of the POOL                    // size(RefSet) =b=b1+b2
RefSet (div1,….,div_b2) = most b2 diverse solution of the POOL                //  selected acc. start or rank dist.
ReserveList=RefSet
bestSol = min{fit(best1), fit(best2),…., fit(bestn)}                          // bestSol is the best solution found so far
bestSolVNSCount = 0                                                           // best solution VNS counter
newSolutionsSwitch=1
alphaCount=0;
iterationCount=0
while newSolutionsSwitch =1
    generate new subsetsk with subset generation method                       // (b2-b)/2 subsets generated
    for each subsetk                                                          // k=1,…., (b2-b)/2
        apply solution combination method and improvement method
    end
    newSolutionsSwitch =0
    update POOL
    update ReserveList
    apply reference set update method and select b1 best solution
    if  min{fit(best1), fit(best2),…., fit(best_b1)} < bestSol
            bestSol = min{fit(best1), fit(best2),…., fit(best_b1)}
            bestSolutionVNSCount=1;                                           // VNS counter
            αCount=0;
            α=α0;              // α0 is the initial α value at time zero
    else
            if bestSolutionVNSCount >VNS activation threshold                 // threshold is a predefined fix value
                αCount= αCount+1                                              // α counter
            end
            bestSolutionVNSCount= bestSolutionVNSCount+1
    end
    if bestSolutionVNSCount = VNS activation threshold
            for all besti Є RefSet
                apply VNS and obtain sol'n best_i^VNS
                if fit (best_i^VNS)< fit (besti)
                    for all besti Є RefSet
                        if best_i^VNS Є RefSet then discard best_i^VNS else replace besti with best_i^VNS
                    end
                end
            end
            if min{fit(best1), fit(best2),…., fit(bestb)} < bestSol
                bestSol = min{fit(best1), fit(best2),…., fit(bestn)}
                bestSolutionVNSCount=1;
                αCount=0;
                α=α0;                                                         // α0 is the initial α value at time zero
```

Figure 4.8. The pseudo code for HCSS approach - SMTWT

Continued

```
            else
                bestSolutionVNSCount= bestSolutionVNSCount+1
                αCount= αCount+1
             end
    end
    apply distance measurement method                                   // start or rank distance
    select diverse solutions from POOL and add them to RefSet
   if αCount>0
            update α                                                    // acc. to one of the alpha strategies
    end
    iterationCount= iterationCount+1
    if iterationCount < = stopping threshold                            // fixed iteration criterion  **
            newSolutionsSwitch =1
    end
    if bestSolutionVNSCount < = stopping threshold                      // dynamic stopping criterion **
            newSolutionsSwitch =1
    end
end
** executed interchangeably
```

Figure 4.8. The pseudo code for HCSS approach – SMTWT (continues)

4.2. HCSS Approach to PMTT

The parallel machine total tardiness problem (PMTT) can be defined as the scheduling of *n* jobs on *m* continuously available identical parallel machines aiming to minimize total tardiness. It differs from the single machine total weighted tardiness problem, PMTT involves both a sequence problem and an allocation problem. A job should be assigned to the right machine and be processed at the right time on that machine in order to achieve minimum total tardiness value.

Our HCSS approach is initially designed for SMTWT problem. Its solution encoding, initial solution generation methods, solution combination methods and VNS are unique techniques that are built based on the single machine scheduling problem. Hence, some essential modifications are performed to adapt it to PMTT problems. With the proper adjustments, refashioned SS covers all the necessities required to search the more complex solution space of PMTT and provide feasible elite solutions. These minor fine-tunings do

not affect the basic idea and the methodology behind HCSS approach. In the forthcoming sections, instead of considering all the steps of algorithm, we only focus on the altered components of our HCSS algorithm.

4.2.1. Solution Encoding Scheme

In our real encoded vector representation, a solution string contains only the start times of jobs and the fitness value. It is a valid encoding for a single machine problem, because the sorted start times define the required dispatching order on a single machine and the fitness can easily be evaluated accordingly. Considering the PMTT problem, the same representation may state which job is processed earlier but it implies no information about on which machine the job will be processed. Therefore, the solution encoding is changed by inserting additional entities involving the machine indices. For an *n* job parallel machine problem, our vector becomes < 2*n*+1> array where first *n* entities denote the start times, second *n* entities denote the machine indices and last entity denotes the total tardiness value. As shown in Figure 4.9, *job #1* is processed on *machine #1* at time 214.81, *job # j* is on *machine #2* at time 257.89 and finally *job #n* is processed on *machine #1* at 83.42.

JOB # 1	*JOB # 2*	*....*	*JOB # J*	*....*	*JOB # N*	*JOB # 1*	*JOB # 2*	*....*	*JOB # J*	*....*	*JOB # N*	*FIT.*
214.81	128.98	....	257.89	....	83.42	1	1	....	2	**....**	1	10.38

Figure 4.9. Solution encoding for PMTT problem

In order to evaluate total tardiness value of a solution, the jobs are grouped with respect to their machine numbers. Then, the jobs on the same machine are sequenced in an increasing order of their start times. Having both of the allocation and the sequence, the fitness value can be easily computed.

4.2.2. Initial Solution Generation Method

Three different initial solution generation methods are developed for the PMTT problem, namely EFT method, EDD method and multi-rule method. All of these methods take their names from the dispatching rules used to schedule jobs.

In Early Finish Time (EFT) Method, each job is assigned to the machine which can complete it earlier. Initially, a temporary start time is generated for each job using the formula;

$$st_j = r_j + (d_j - r_j) \text{ x rand} \quad (4.5)$$

where st_j, r_j, and d_j denotes start time, ready time and due date of job j, respectively and *rand* is a random number between 0 and 1. Then these start times are sorted in order of non-decreasing times and a sequence is obtained. Beginning with the first job in the sequence, for each job; its completion times on each machine is computed and it is dispatched to the machine with the earliest completion time. Finally, the total tardiness value is calculated and our initial trial solution becomes ready with its start times, machine numbers and fitness value.

```
Start with P = Ø, where P denotes initial population and x denotes an initial trial sol'n Є P.
while size(P) < P_size
    for j = 1,…..,n      // n denotes the # of jobs
        st_j = rt_j + (dt_j – rt_j) * rand          // st_j -start time, dt_j- due date, rt_j- ready time of job j
    end
    sort j's in order of non-decreasing st_j and obtain a sequence q contain sorted j's
    for k =1,…..,n
        compute completion time of q(k) on each machine_i          // i =1,…., m m is the # of machines
        select the machine_i with earliest finish time
        assign job_q(k) to selected machine_i
        update start, tardiness, and completion time
    end
    sum all tardiness and calculate fitness
    x' = [start times of job_j + fitness]                                                    // j=1,…..,n
    if fit(x') ≠ any of fit (x),  x Є P the x' is added to P, x'→x, otherwise discard x'
    P=P U x
end
```

Figure 4.10. The pseudo code for EFT initial solution generation method

In Earliest Due Date (EDD) Method, initially jobs are assigned to the machines randomly. Jobs are then sorted according to their due dates. Job with an earlier due date gets the precedence to be processed earlier on its machine assigned. By dispatching jobs randomly to the machines, we convert the parallel machine problem into *m* independent

single machine problems, where *m* denotes the number of machines, and EDD rule is used to sequence jobs. The total tardiness and start times values of jobs are found separately for each machine and then these data are collected together to form the initial trial solution.

Multi-Rule Method is an advanced version of EDD Method. In addition to EDD rule, SPT and XR&M (Morton and Pentico 1993) rules are also utilized to order jobs on a given machine. EDD rule, as previously mentioned, arranges jobs according to their due dates. Shortest Process Time (SPT) Method ranks the jobs in an order of non-decreasing process times and finally XR&M is a modified version of R&M priority rule which is discussed in single machine problem. R&M developed for SMTWT problem, uses weights to evaluate *twt* value and all jobs are ready at time zero. However, in PMTT problem, there are no related weights and the jobs have distinct ready and set-up times. Therefore necessary adjustments are conducted to adapt R&M formulas to PMTT problem. In first place, the slack time S at time *t* is found by;

$$S_j = d_j - r_j - p_j - t \tag{4.6}$$

where subscript *j* indicates the job *j* and *t* indicates the completion time of the job *j*-1 on the same machine. The idle priority rule for a given job *j* is evaluated using the formulas;

$$(\pi_j)_{idle} = \pi_j [1 - B (r_j - t)^+ / p_{av}] \tag{4.7}$$

where r_j is ready time of job *j*, p_{av} is the average processing time of all jobs assigned to the same machine and

$$\pi_j = (1 / p_j)[\exp\{-(S_j)^+ / p_{av}\}] \tag{4.8}$$

$$B = 1.3 + \rho \tag{4.9}$$

ρ is the average utilization of the machine which is found by

$$\rho = \text{total process time assigned} / \quad \text{current time} \tag{4.10}$$

At time zero, $(\pi_j)_{idle}$ value of all jobs are evaluated and the job with the highest $(\pi_j)_{idle}$ is the first to be scheduled. Afterward, according to the completion time of selected

job, the priorities of remaining jobs are updated and a new job is selected depends on these values. This iterative process continues until all jobs are sequenced.

Since we assign jobs to machines randomly, the PMTT is transformed into single machine problems aiming to permute jobs in order to minimize total tardiness. Dealing with these single machine problems, three dispatching rules; EDD, SPT and XR&M are applied to same machine respectively. The obtained sequences with resulting fitness values are compared and the sequence with the smallest fitness value is chosen for the considered machine. For example, in a two machine problem, the three rules are applied for both *machine #1* and *machine #2*. For the first machine, EDD performs better and for the second, XR&M outperforms the others. Then our final trial solution in constructed by using the EDD sequence for *machine #1* and XR&M sequence for *machine #2*. The total fitness is the sum of tardiness resulted from EDD and XR&M dispatching. The performances and diversities of EFT, EDD and multi-rule generated initial solution sets are judged in experimental results section.

4.2.3. Solution Combination Method

In PMTT, a modified two-phase BLX-α operator is employed to generate new trial solutions. During the first phase, the start times of the offspring are generated as performed in the SMTWT. The requirement of second phase arises when a machine number is needed to be assigned for each job. The two parents have the data of appointed machine numbers that is stored in their chromosomes, which should be transferred somehow into newly generated offspring. Second phase uses a conditional random assignment procedure to select appropriate machine for each job by looking at the genetic heritage of parents. If a job is processed on the same machine in both of the parent solutions then this machine processes the corresponding job of the offspring. If the two machines are different in the parent solutions for the same jobs, then the job is assigned to a machine that is randomly selected considering the machines of the parent solutions. For example, for job *j*, it is processed on *machine #1* in one solution and on *machine #2* in another solution. Considering the offspring generated from these solutions, the machine for job *j* will be either *machine #1* or *machine #2*. This selection is performed randomly. After finding start

times and assigning machines for each job, the total tardiness is evaluated and the new trial solution is created.

```
Start with P = Ø, where P denotes initial population and x denotes an initial trial sol'n Є P.
while size(P) < P_size
    randomly assigned each job_j to a machine mch_i        // i=1,...., # of machines (m), j=1,....,# of jobs (n)
    for each mch_i
        sort all job_j Є mch_i according to EDD                                   // Earliest due date
        schedule all job_j on mch_i and evaluate start times and fitness value
        store st  and fitness value in a string x_EDD = [st_1,….,st_n, fit]       // st_j is start time for job j
        sort job_j Є mch_i according to SPT                                       // Shortest process time
        schedule all job_j on mch_i and evaluate start times and fitness value
        store start times and fitness value in a string x_SPT = [st_1,….,st_n, fit]   // x_SPT is partial schedule
        remain jobs = all job_j Є mch_i
        while size(remain jobs) ≠ 0
            for each job_j Є remain jobs
                update slack time, π_j, ρ                 // π_j is the priority of job_j , ρ is avg. mach. util.
                calculate (πj)_idle                                               // (πj)_idle is the idle priority
            end
            select job_j with the max (πj)_idle
            calculate its start time, completion time and tardiness on mch_i
            delete job_j from remain jobs
        end
        sum all tardiness and obtain fitness
        store start times and fitness value in a string x_XRM = [st_1,….,st_n, fit]   // x_XRM is partial schedule
        select schedule_best with min fitness     // x_EDD, x_SPT or x_XRM is selected as schedule_best
    end
    x' = [(schedule_best)_mch1 + (schedule_best)_mch2 +…….+(schedule_best)_mchm]
    if fit(x') ≠ any of fit (x),  x Є P the x' is added to P x'→x, otherwise discard x'
    P=P U x
end
```

Figure 4.11. The pseudo code for multi-rule initial solution generation method

4.2.4. Variable Neighborhood Search

VNS applied to SMTWT problem is quite simple. It concerns the sequence of jobs on a single machine. So, the neighborhoods of a current solution can easily be generated by swap moves that provide different permutations. Consequently, the final fitness value is updated by evaluating these neighborhood sequences. On the other hand, PMTT problem

consists of both allocation and scheduling of *n* jobs on parallel machines. Therefore, in order to search the neighborhoods of a current solution, VNS should generate candidate solutions by applying both machine interchanges and swap moves. For this reason, three different type VNS approaches are developed and implemented to elite solutions one after other at times determined by best solution VNS counter.

In the first type VNS (*type #1*) technique, the sequence of a selected elite solution is recorded and kept fixed during the entire neighborhood search. Then this sequence is divided into subsequences containing same number of jobs defined by VNS depth. Starting with the first subsequence, all possible machine assignments are listed for this group's jobs. As an example, considering a two machine problem with VNS depth equals to three, the all possible assignments can be stated as (111), (112), (121), (211), (221), (122), (212) and (222), where (112) denotes that the first job of the selected group is processed at machine #1, and the remaining jobs are at machine #2. Later, all these alternative assignments for first three jobs are investigated without changing the initial sequences and other machine assignments for the jobs stay out of this group. The alternative with the smallest fitness is selected and the current machine assignment is updated accordingly. When all the groups are treated in the same way and the assignments are revised, the total tardiness value of the neighborhood solution is compared with the current solution. If there is an improvement, neighborhood solution becomes the new current solution and VNS depth is decreased one level. Otherwise, the depth is increased one level and VNS repeat the previous steps for the new neighborhoods of the current solutions. See Appendix E.

In the second (*type #2*) and third type (*type #3*) VNS techniques, the methodology employed to construct neighborhood of a current solution is exactly the same that is used in SMTWT. Without considering machine allocations, the initial sequence of an elite solution is divided into subsequences and each subsequence involves its permutation alternatives. The neighborhoods are generated using these permutations that alter the initial sequence. The only difference arises at the evaluation step of the fitness value. In SMTWT problem, the order of jobs contains all the data need to calculate *twt* value. On the other hand, only a sequence of job indices means nothing for a parallel machine scheduling problem because it does not contain any information about machine assignments.

Therefore, two simple dispatching rules are utilized to cover required machine indices for each job.

The type #2 VNS technique employs an early finish time rule which starts with the initial job of the defined array and computes its completion times on each machine separately. These values are then compared and the machine with the earlier completion time is selected. After assigning the first job, the next one is dispatched to a machine by repeating the same computations. This scheduling continues until all jobs have a machine to be processed on. And finally, the total tardiness is evaluated and the neighborhood solution is generated according to the obtained job sequence and machine allocations. See Appendix E.

The type #3 VNS technique uses a recursive rule that operates in a one-way track fashion over branch alternatives. In this method, start times play the key role to decide a machine for a given job. Jobs are allocated according to their positions stated by the sorted sequence. At each step of allocation, a start time corresponding to a job is compared with the finish time of the machines where the finish time denotes the completion time of previously assigned job on that machine. If the start time of a selected job indicates an instant that is before the both machine finish their duties, then the selected job is assigned to a machine with the earliest completion time. Else if, the start time is earlier than one finish time but not the other, then the job corresponding to that start time is scheduled on the earlier machine. Finally, if there is a slack time between the finish time and the start time for both machines, then the job can be assigned any of the machines. Thus, recursive rule constructs two branches at that node which contain each assignment alternatives. When all the jobs are dispatched and their alternative branches are built, the tardiness value at each node is evaluated. Starting from the first node and moving forward to the last node through generated branches, total tardiness value is calculated. In addition to that, this one-way track move provides the required assignment sequence for the neighborhood solution. The assignment with the smallest total tardiness becomes the fitness of sequence that is generated initially as the candidate solution of the initial solution. See Appendix E.

The other features of VNS that are not mentioned in this section, is same as in SMTWT problem. Different from the single machine problem, our best solution VNS

counter triggers three different type VNS techniques. It initially triggers first type VNS technique which generates neighborhoods based on machine interchanges for a fixed sequence of jobs. If the current solution can not be improved and counter reaches to the second threshold, it executes the second type VNS where the evaluations of neighborhoods are done by using EFT rule. Depending on the presence of an improvement, counter either implements the first type VNS or the third type VNS technique. If HCSS or previous VNS attempts fails to upgrade the best known solution, third type VNS is executed to search new neighborhoods attentively while considering all schedule alternatives related to them. Third type VNS technique is an intensive search method and it becomes highly time-consuming when the number of jobs is increased. Therefore, it is applied at the final stage like a last bullet to the target. If the bullet hits the target, the counter reset itself and the search starts from the beginning with a superior solution. Otherwise, HCSS continues for a fixed iteration and stops if no improvement is achieved. The performance comparison between three type VNS and single type VNS technique used in SMTWT problem will be performed in experimental result section and the best method will be selected for our final HCSS approach.

```
create initial solution set P
POOL = P
Select b1 best solutions and b2 diverse solutions of RefSet from  the POOL
ReserveList=RefSet
bestSol = min fitness value RefSet solution
bestSolVNSCount = 0
newSolutionsSwitch=1
alphaCount=0;
iterationCount=0
while newSolutionsSwitch =1
    generate new subsetsk with subset generation method
    for each subsetk
        apply solution combination method and improvement method
    end
    newSolutionsSwitch =0
    update POOL
    update ReserveList
    apply reference set update method and select b1 best solution
    update bestSol, bestSolutionVNSCount, αCount, α
    // (activation threshold of type #1 MVNS < type #2 MVNS < type #3 MVNS)
```

Continues

Figure 4.12. The pseudo code for HCSS approach – PMTT

```
    if bestSolutionVNSCount =activation threshold for  type #1 MVNS
            for all best_i Є RefSet
                apply type #1 MVNS and obtain sol'n best_i^VNS
                update RefSet with best_i^VNS  if necessary
            end
            update bestSol, bestSolutionVNSCount, αCount, α
    end
    if bestSolutionVNSCount =activation threshold for type #2 MVNS
            for all best_i Є RefSet
                apply type #2 MVNS and obtain sol'n best_i^VNS
                update RefSet with best_i^VNS  if necessary
            end
            update bestSol, bestSolutionVNSCount, αCount, α
    end
    if bestSolutionVNSCount =activation threshold for type #3 MVNS
            for all best_i Є RefSet
                apply type #3 MVNS and obtain sol'n best_i^VNS
                update RefSet with best_i^VNS  if necessary
            end
            update bestSol, bestSolutionVNSCount, αCount, α
    end
    apply distance measurement method
    select diverse solutions from POOL and add them to RefSet
    if αCount>0
            update α =0.5+(Δ* αCount)
    end
    if bestSolutionVNSCount < = stopping threshold                    // dynamic stopping criterion
            newSolutionsSwitch =1
    end
end
```

Figure 4.12. The pseudo code for HCSS approach – PMTT continues

5. NUMERICAL STUDIES

This chapter provides the details of experimental procedure and different strategies applied to tardiness related scheduling problem sets together with the numerical results. The algorithm is implemented on MATLAB® (MathWorks, 2006) which is a high-performance language for technical computing. The name MATLAB® stands for matrix laboratory. It is an interactive system whose basic data element is an array that does not require dimensioning. This allows solving many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran. The solution representation of our HCSS approach consists of vectors and arrays. Therefore, the basic features of MATLAB® provides the necessary computing language infrastructure for our meta-heuristic approach and let the incorporation of further strategies composed in different compilers within MATLAB®. Both parallel and single machine total tardiness/total weighted tardiness problems in their most generic forms, i.e. with distinct ready times, processing times, due dates and sequence dependent setup times are imported into the software as matrices and can be processed by unique codes special to matrix operations.

The several strategies developed for HCSS approach are tested via MATLAB® and the results are reported in the following sections. The solution strategy to be employed for each problem can be specified by the user just selecting a combination of methods implemented for parallel or single machine total tardiness/total weighted tardiness problems separately. Extensive experimentation is performed on the HCSS approach developed in this thesis. The experiments are conducted on a AMD Athlon 64 – 1.81 GHz CPU with 512 MB RAM.

5.1. Problem Set for SMTWT

The problem set used for experimentation consists of scheduling problems with 40, 50, 100-jobs, which is developed and tested by Crauwels et al.(1998). A total of 125 test instances are available for each problem size n=40, n=50 and n=100. The instances were

randomly generated as follows: For each job *j (j=1,...,n)*, an integer processing time p_j was generated from the uniform distribution [1,100] and integer processing weight w_j was generated from the uniform distribution [1,10]. Instance classes of varying hardness were generated by using different uniform distributions for generating the due dates. For a given relative range of due dates, *RDD* (*RDD*=0.2, 0.4, 0.6, 0.8, 1.0) and a given average tardiness factor *TF* (*TF*=0.2, 0.4, 0.6, 0.8, 1.0), an integer due date d_j for each job *j* was randomly generated from the uniform distribution [$P(1\text{-}TF\text{-}RDD/2)$, $P(1\text{-}TF\text{+}RDD/2)$], where $P = \Sigma_j p_j$ for *j (j=1,...,n)*. Five instances were generated for each of the 25 pairs of values of *RDD* and *TF*, yielding 125 instances for each value of *n*. These instances are available in the OR library run by (Beasley, 2006), which is a collection of test data sets for a variety of Operation Research problems.

Crauwels et al. (1998) develop a Branch and Bound algorithm to attack *n*=40 and *n*=50 instances. They manage to solve 124 out of the 125 instances for *n*=40 and 103 problems out of 125 instances for *n*=50 with a time limit of two minutes for each instance whereas *n*=100 instances were abandoned due to the anticipation of extremely high computational times. The unsolved 40-job problem is number 19, and 50-job problems 11, 12, 14, 19, 36, 44, 66, 87, 88 and 111 remain unsolved. The optimal solutions for 124 of the 40-jobs problem instances and 115 of the 50-job instances are given in the OR library. As for, 100-jobs problem instances, the best known solution values reported by Crauwels et al. (1998) and Congram et al. (2002) are available. Since, the solutions not known to optimality have not been improved further with respect to their best-known values, there is a strong evidence that they are actually the optimal solutions. Appendix F.1 presents the optimal and best known solutions for 40, 50 and 100-jobs problem instances as reported in the OR library.

5.2. Problem Set for PMTT

The problem set used for experimentation consists of parallel machine scheduling problems of 40 and 60 jobs, developed and tested by Sivrikaya-Şerifoğlu and Ulusoy (1999). A total of 20 test instances are available for each problem size, i.e. $n = 40$, $m = 2$ and 4 and $n = 60$, $m = 2$ and 4 where *n* denotes the number of jobs and *m* denotes the number of machines.

The instances were randomly generated as follows. It has been assumed that machines belong to one of two different types, which have the same characteristics except that they have different processing times. Type II machines represent an older technology. The processing time of a job on a Type II machine is 10-20 per cent greater than its processing time on a Type I machine. Similarly, setup times on a Type II machine are 20-40 per cent larger than the corresponding setup times on a Type I machine. Processing times of job j on a Type I machine, p_j^I follow the uniform distribution U [4,20]. To generate the processing time of job j on Type II machine, which is denoted as p_j^{II}, a multiplier is chosen randomly from [1.10, 1.20] and is applied to the processing time of job j on the Type I machine.

Setup times on Type I machines, cited as a^I, are taken to be uniformly distributed with U [1, A_{max}] where two levels of A_{max} are utilized in this study. Again a multiplier chosen from [1.20 1.40] is employed to compute the setup times on Type II machine. Ready times are assumed to follow the uniform distribution U [0, R_{max}], where R_{max} is the maximum ready time. Here, $R_{max} = \lceil (p_{avg}^{II} + a_{avg}^{II})/(N/M\text{-}1) \rceil$ where $\lceil x \rceil$ is the smallest integer greater than or equal to x, and p_{avg}^{II} , a_{avg}^{II} are the average processing time and average setup time on machine Type II respectively. The due date of job j is taken to be the sum of its ready time, processing time on the Type II machine, maximum time to setup a Type II machine for the processing of job j and a slack value. The slack value S is defined as the sum of mean values of processing and setup times on a Type II machine:

$$S = p_{avg}^{II} + a_{avg}^{II}. \tag{5.1}$$

Due dates are computed according to the formula:

$$d_j = r_j + \max_i a_{ij}^{II} + p_{ij}^{II} + S \tag{5.2}$$

Bilge et al. (2004) apply a deterministic TS algorithm and obtain high quality solutions with respect to earlier results from the literature for the same problem set. However, Anghinolfi and Paolucci (2006) represent a hybrid meta-heuristic approach that integrates TS, SA and VNS and achieve some superior results than the ones found by Bilge et al. Both sets of best known solutions cited by the authors are given in Appendix F.2.

5.3. The Experimental Procedure for the HCSS Approach

The performance measure used for this study is the average percentage of deviation from the best value known in the literature. Hence, in all the experimental results presented in the figures, the percentage deviation for an instance is calculated by using the formula:

$$\text{per cent dev} = [(\text{Best} - \text{Best-known to literature})/ \text{Best-known to literature}] \times 100 \quad (5.3)$$

where Best-known to literature is the best known solution reported in OR library for the problem set and Best is the best solution obtained by the HCSS. Then the average of total percentage deviations for a problem set is computed as

$$\text{avg. per cent dev.} = \Sigma_i \, (\text{per cent dev.})_i \, / \, \text{total number of instance} \quad (5.4)$$

where *i* denotes the number of an instance. In each experimentation phase, only aggregated results are used for decision-making purposes. Therefore, only those aggregate forms of results are presented for the entire set of problems, rather than presenting individual results for each instance. The performance evaluation is based on the average per cent deviation from the optimal (or best known value to the literature) of all the instances of considered problem sets. Individual superior results for some instances do not affect the final decision. In other words, each candidate strategy is evaluated in terms of aggregate performance quality instead of individual success per problem instance.

For SMTWT problems, mostly 50-job instances are considered. While making a crucial decision concerned with the methodology of HCSS approach, the results found for 50-job instances are supported by either 40-job, 100-job or both sets. Hence, more reliable decisions can be made. This is because in some cases the difference between avg. per cent deviations of two compared candidate methods is negligible when 50-job instances are taken into account. On the other hand, the difference can become more significant for 100-job problem set. Based on this argument, instead of evaluating a strategy with respect to single set of results, more data are considered to arrive at a more robust final form of our approach.

For PMTT problems, two-machine problems are usually harder than their four-machine counterpart. For this reason, the experimentation is conducted with 40-job two-machine instances. Hence, in the experimentation phase, only 40-job two-machine problem average per cent deviation results are reported and the decisions are made after comparing these results. The final form of the algorithm obtained in this manner is then employed for the remaining problem sets for final results.

Since the total number of strategies to be tested is very large, a sequential experimentation procedure is adopted. First, the HCSS approach is implemented in its most elementary form, which we call the basic algorithm. Then this basic algorithm is developed into its final form by fixing strategies according to the experimentation results. At each experiment, we test one or more parameters/methods and select one (or sometimes more) level that perform at least as good as the others for each instance, and go on with this new form of the algorithm.

Before examining experimental results for SMTWT, the components of our basic HCSS model are introduced briefly. In our basic model, the initial solutions are generated randomly *(rand)*. Initial solution set size is considered as 150 which is consistent with the values given in Herrera et al (2006) and Laguna et al (2006). Hence, the size of the set will not be fine-tuned and kept constant during entire experimentation. The reference set consists of 10 high quality and 10 diverse solutions, as suggested by Laguna et al. (2006). Best solutions are selected from the reserve list *(2-tier)* whereas diverse solutions are chosen according to their start time distances *(start)* to the reference set as described in Section 4.1.3. The solution combination method employs BLX-α operator where α equals to 0.5 *(static)* and the generated trial solutions are stored in the pool which has a capacity defined as *pool size*. The initial pool size is considered as 100, which means 90 of the generated trial solutions are not accepted to the pool depending on their poor fitness values. In our elementary HCSS model, no intensification strategy such as VNS is performed; different types of VNS methods will be introduced in further sections of experimentation. Finally, the stopping criterion in the basic algorithm is set to 150 iterations. The used components and default parameter values of our basic HCSS algorithm is presented in Table 5.1. Until the otherwise is stated, these components and parameters are implemented in our algorithm at the experimentation stage.

Table 5.1. Basic HCSS algorithm

	BASIC HCSS
REFERENCE SET	
UPDATE METHOD	2-tier + start
# of BEST	10
# of DIVERSE	10
INITIAL SOLUTION SET	
GENERATION METHOD	RANDOM
# of INITIAL SOL's	150
POOL	
SIZE	100
SOLUTION COMBINATION METHOD	
ALPHA STRATEGY	STATIC
ALPHA VALUE	0.5
INTENSIFICATION STRATEGY	
TYPE OF VNS	NO
SEARCH DEPTH	X
STOPPING CRITERION	
TYPE	FIX
# of ITERATIONS	150

For every experiment conducted, a figure and a table are given together to report the results of this experiment and to summarize the components of HCSS which is implemented in the experiment. The table is divided into columns and each column represents a component of our model. An illustration of our basic model is given in Table 5.2.

Table 5.2. Components of basic HCSS

INISOL	**REFSET**	**DISTANCE**	**VNS**	**DEPTH**	**POOL**	**ALPHA**	**STOP**
RAND	2-TIER	START	NO	X	100	0.5	150

where INISOL denotes the initial solution generation method, REFSET denotes reference set update method, DISTANCE denotes distance measurement method, VNS and DEPTH denote intensification strategy and its search depth, POOL denotes pool size, ALPHA denotes α value used in solution combination method and finally STOP defines the stopping criterion. Each time, when a new alternative method or parameter value is introduced, it is symbolized with abbreviations which are referred in brackets. This representation provides a visual simplicity while presenting the internal dynamics of the

algorithm. Considering the figures, the x-axis contains the compared methods or strategies, y-axis defines problem set and z-axis states the average per cent deviations obtained from all instances.

5.4. Numerical Experimentation for SMTWT

As the first step of experimentation, two diverse solution selection methods and three pool sizes are tested by taking all of their combinations. Considering 40-job instances, restricted pool sizes perform better regardless of the distance measurement method. As shown in Figure 5.1, start distance measure with pool size 100 *(start 100)* and pool size 85 *(start 85)* result in 2.11 and 1.62 average per cent deviation values respectively, which are both smaller than the value 2.22 obtained by using start distance measure with no pool restriction *(start all)*. The same situation can be observed between rank distance measure and pool size parameter *(rank 100, rank 85)*. Again, two restricted pool sizes come up with superior performance values. This is expected since; without any restriction, too diverse poor quality solutions are accepted to pool and selected as an element of reference set by one of distance measurement methods. Therefore, a poorly constructed reference set generates weak offspring that do not yield satisfactory results. Investigating all problem sets, rank 100, start 100 and start 85 are considered as the most promising distance measure-pool size combinations.

The next step is to determine search depth for our intensification strategy (VNS). Middle VNS (MVNS) is employed with basic scatter search and two different search depths; 3-4-5 and 4-5-6 are tested for 50-job instances as described in Section 4.1.6. From the Figure 5.2, it can be seen that MVNS 4-5-6 outperforms MVNS 3-4-5 regardless of the distance measurement method and pool size combination. Although it is not shown here, two compared search depths are chosen among a wide range of depths varying from 2-3-4 to 6-7-8. The depth 3-4-5 is the best performed one among the fast VNS methods, whereas depth 4-5-6 is the fastest when VNS with high quality outcomes are considered. The comparison between these two states that MVNS 4-5-6 has a remarkable superiority over

Table 5.3. HCSS for SMTWT - pool sizes and distance measurement methods

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
RAND	2-TIER	START	all, 100, 85	NO	X	0.5	150
RAND	2-TIER	RANK	all, 100, 85	NO	X	0.5	150

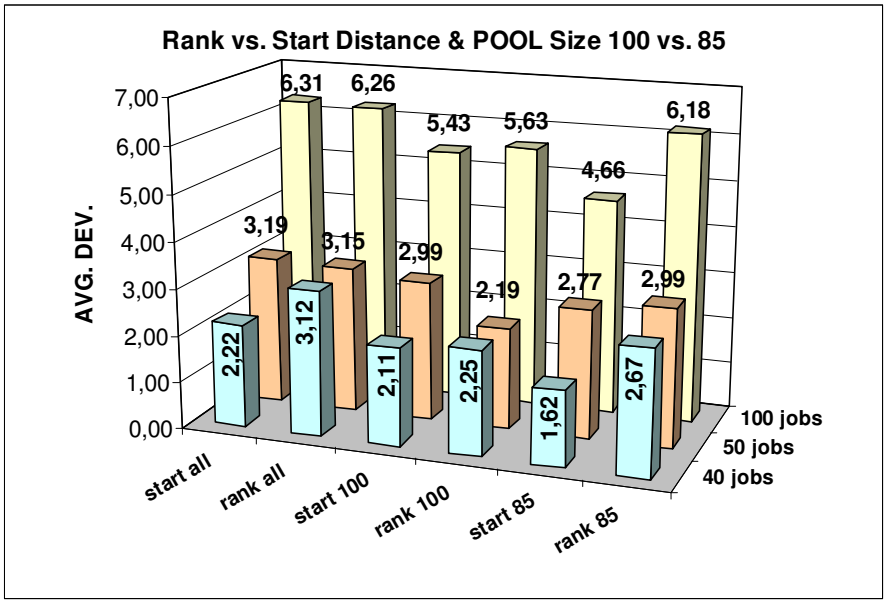

Figure 5.1. Experimental results for pool sizes and distance measurement methods

its faster alternative. If the difference were not so significant, the faster VNS depth 3-4-5 could have been preferred. Hence, during our remaining experimentation, search depth 4-5-6 is used for our variable neighborhood search.

After deciding the search depth, two different VNS strategies are applied to 50-job 100-job problem sets. In the first strategy, a final VNS is executed only on the best solution found so far as soon as the stopping criterion – 150 iterations is met whereas for the second strategy a middle VNS is implemented when the non-improving iterations counter reaches a certain threshold. This threshold is defined as 30 consecutive non-improving iterations. Considering 50-job instances, there can be observed a clear improvement after applying an intensification strategy; but no clear selection can be made between FVNS and MVNS. This is because; the performances shown in Figure 5.3 are close to each other. In order to avoid an improper conclusion, a more challenging problem set consisting of 100-jobs is

Table 5.4. HCSS for SMTWT - comparison of search depths

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
RAND	2-TIER	START	85	MVNS	3-4-5	0.5	150
RAND	2-TIER	RANK	100	MVNS	3-4-5	0.5	150
RAND	2-TIER	START	85	MVNS	4-5-6	0.5	150
RAND	2-TIER	RANK	100	MVNS	4-5-6	0.5	150

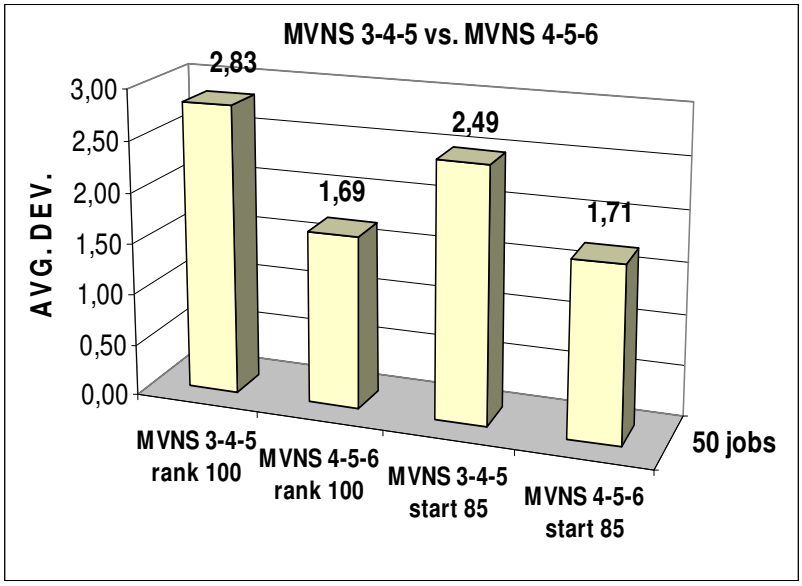

Figure 5.2. Fine-tuning for search depth parameter

attacked by MVNS can be clearly proven when combined with both rank 100 and start 85. Hence, MVNS 4-5-6 is selected as the intensification strategy for HCSS approach.

Table 5.5. HCSS for SMTWT - comparison of different intensification strategies

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
RAND	2-TIER	START	85	FVNS	4-5-6	0.5	150
RAND	2-TIER	RANK	100	FVNS	4-5-6	0.5	150
RAND	2-TIER	START	85	MVNS	4-5-6	0.5	150
RAND	2-TIER	RANK	100	MVNS	4-5-6	0.5	150
RAND	2-TIER	START	85	NO	X	0.5	150
RAND	2-TIER	RANK	100	NO	X	0.5	150

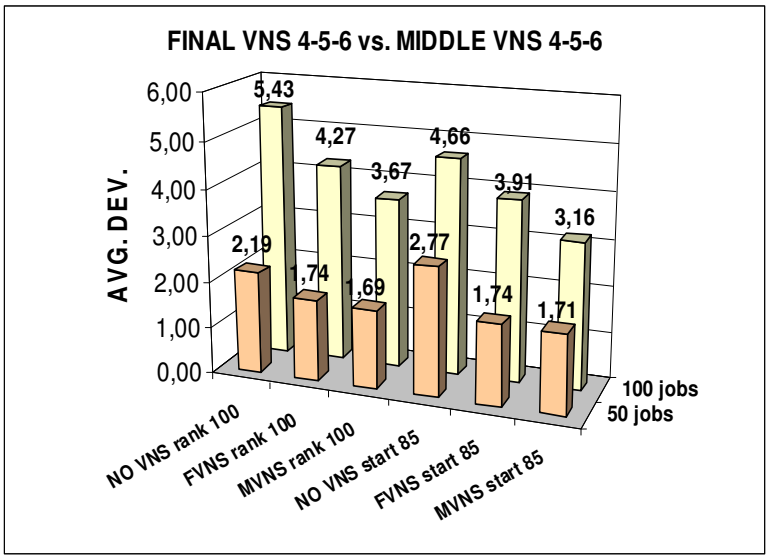

Figure 5.3. Effects of intensification strategies

So far, the initial population used in HCSS is randomly generated. From now on, we discuss the performance of two new population generations approaches. For the experiments performed at this stage, all the new population are formed by either seeding the randomly generated population with some selected good solutions found at the end of various list scheduling heuristic *(SEED-10)* or by seeding the random population with all good solutions obtained by same heuristic *(SEED-ALL)*. The list scheduling heuristics employed are EDD, SPT, WSPT and R&M +. A single seed solution is provided from each of the heuristics except R&M +. R&M + provides up to 36 different elite solutions by varying *k* value. Either all of these solutions are accepted to initial solution set or best 10 of them are selected as described in Section 4.1.2.

Based on the results presented in Figure 5.4, seeded initial solution set containing best 10 R&M solution with pool size 85 performs better for both 40-job and 100-job instances. Focusing on 40-job problem set, seeded initial solution set *SEED-ALL* do not yield satisfactory result as compared to *SEED-10* or randomly generated population . The main reason behind this situation is the selection of almost identical individuals to the initial population. (Trial solutions with different start times but same fitness value can be observed in the initial population at the same time; because the selection procedure only filters the identical individuals.) Similar individuals later dominate the reference set and

directly affect the next generations formed from their linear combinations. Hence, insufficient diversity in the population avoids HCSS to explore different promising regions and the search cycles around the same local optima.

Table 5.6. HCSS for SMTWT - comparison of different initial solution generation methods

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
RAND	2-TIER	START	100,85	MVNS	4-5-6	0.5	150
SEED-ALL	2-TIER	START	100,85	MVNS	4-5-6	0.5	150
SEED-10	2-TIER	START	100,85	MVNS	4-5-6	0.5	150

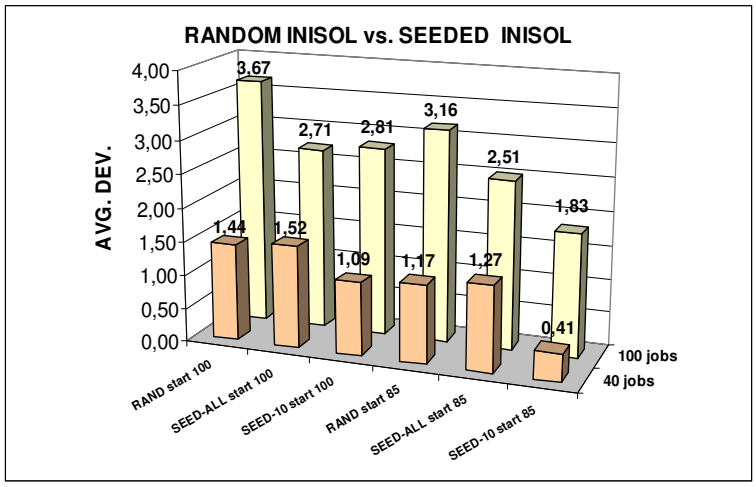

Figure 5.4. Effect of initial solution generation methods

A screening mechanism is developed to overcome the diversity problem in the initial population. This mechanism removes one of two trial solution with the same tardiness value. By doing so, it is guaranteed that every screened solution has a different sequence of jobs which reflects an important indicator of diversified population. In the experimentation, screening is only applied to seeded initial solution set with best 10 R&M solution *(SEED-SCR)* and the results are summarized in Figure 5.5. It is obvious that screening improves both diversity and quality of initial population; thus affects the efficiency of search.

Previously, MVNS is applied only to first three best solutions stored in the reference set. These best solutions are directly transferred from the reserve list where best

Table 5.7. HCSS for SMTWT - effect of screening mechanism

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
RAND	2-TIER	START	100,85	MVNS	4-5-6	0.5	150
SEED-10	2-TIER	START	100,85	MVNS	4-5-6	0.5	150
SEED-SCR	2-TIER	START	100,85	MVNS	4-5-6	0.5	150

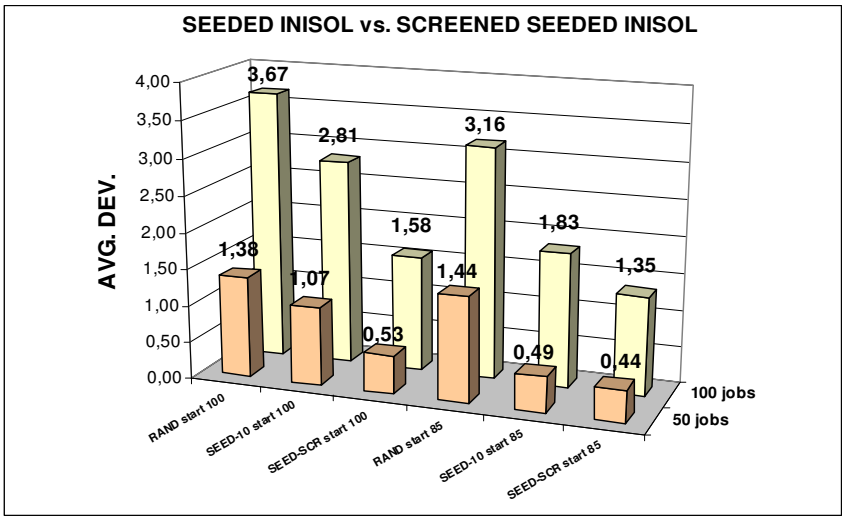

Figure 5.5. Effect of screening mechanism

found solutions are kept. Without considering other elite solutions, our intensification strategy becomes a partial exploitation mechanism that neglect promising neighborhoods of other best solutions. As mentioned before, the performance of a solution is evaluated by only looking at its fitness value. A solution, which is considered as poor, can be converted in to a better solution by just swapping two jobs' positions. Therefore every solution is assumed to have a potential local optimum hidden at its neighborhood. Based on this assumption we developed a comprehensive MVNS and explore the neighborhood of all elite solutions contained in the reference set *(MVNS ALL)*. The experimentation is performed with a randomly generated initial solution set and the results for 40-job and 50-job instances are presented in Figure 5.6 MVNS ALL with start 85 decreases average per cent deviation from 1.17 to 0.30 for 40-job problem and from 1.71 to 0.47 for 50-job instances. This is a remarkable improvement. Hence, new MVNS method is adapted as our new intensification strategy.

Table 5.8. HCSS for SMTWT - comparison of MNVS

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
RAND	2-TIER	START	100,85	MVNS	4-5-6	0.5	150
RAND	2-TIER	START	100,85	MVNS ALL	4-5-6	0.5	150

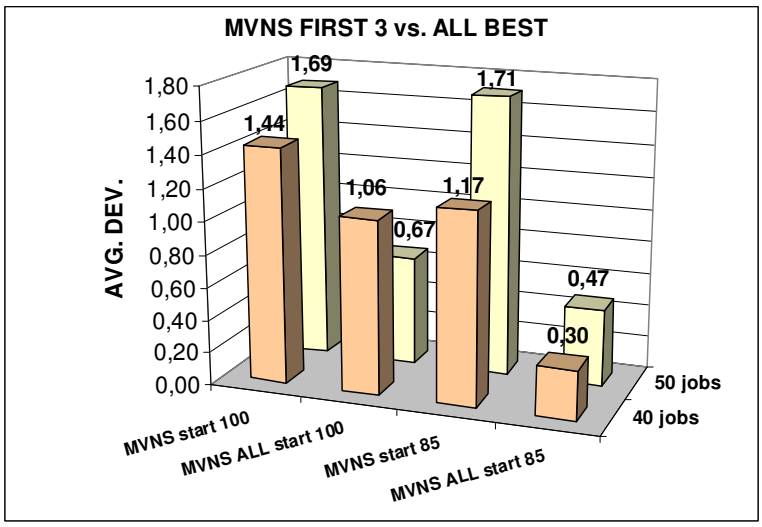

Figure 5.6. Experimental results for compared MNVS methods

At this point, an important question comes to mind, what happen if we design a HCSS which start with a better initial solution set and employs a complete intensification strategy that covers all elite solutions. In Figure 5.7, performances of new intensification strategy, new seeded population with screening and their combinations are shown separately. Based on the results observe from the figure, the new combination shows a perfect harmony for all instances and reduce average per cent deviation value in both cases. Therefore, MVNS with enlarged application area and diversified seeded population are adapted together to our HSCC approach and further experimentations are implemented accordingly.

Table 5.9. HCSS for SMTWT – effect of combined initial solution and MVNS methods

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
SEED-SCR	2-TIER	START	85	MVNS	4-5-6	0.5	150
RAND	2-TIER	START	85	MVNS ALL	4-5-6	0.5	150
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	0.5	150

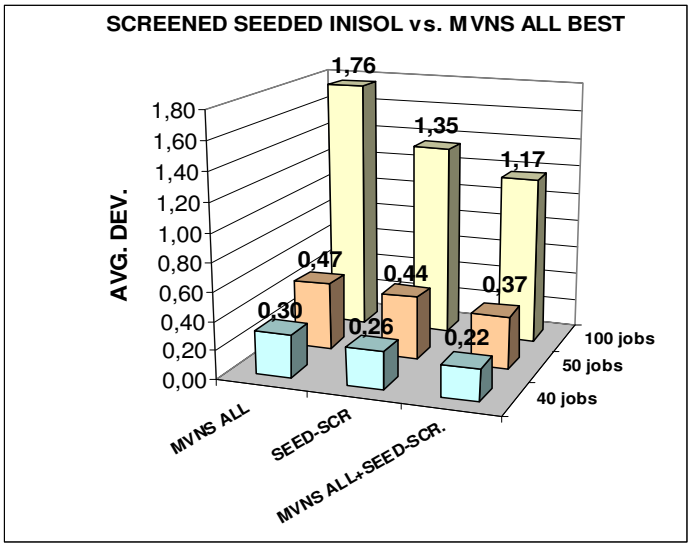

Figure 5.7. Combination of screened initial solution set with new MVNS method

Diversity of the population is the main force behind the HCSS that push it to explore distant promising regions in order to find superior solutions. For that reason, diversification among the individual must be control at every stage of the search. The initial diversity of population is obtained by new screening mechanism, but alone it is not sufficient to keep diversity at desired levels for next generations. As we discussed in Section 4.1.5, solution combination method that utilizes BLX-α operator has the ability to promote diversity of the population. It causes variances of distributions increase while spreading them in this way. BLX-α provides a useful tool to enhance exploration capability. In order to control diversity, α parameter is employed in three different strategies based on dynamically changing α value. Those three strategies are implemented as explained in Section 4.1.7 for all instances and the results are given in Figure 5.8. The strategy that start with α value equal to 0.5 and increases it by 0.005 at the end of each non-improving iteration, performs better than other strategies which are based on either static or dynamic α value.

Before modifying our HCSS approach with dynamic alpha strategy, a fine-tuning step is executed for the increment values (Δ); 0.005, 0.01, 0.025 and 0.05. These increments are tested on 50-jobs and 100-job instances and the results are reported in Figure 5.9. It is obvious that the smallest increment 0.005 outperforms its candidates. An

interesting result is observed for 100-job problem where the average per cent deviations are too high for 0.025 and 0.05. This shows that large α values used in solutions combination method generates too diverse solutions that lead the search to poor and low quality regions. As a result, α parameter should be keep below a certain limit and fast convergence to this limit should be prevented. Otherwise, the diversification strategy becomes an inaccurate and blind exploration tool.

The next step is to compare fixed iteration and dynamic iteration stopping criteria. As described in Section 4.1.8., fixed iteration criterion terminates HCSS at the end of 150 iterations; on the other hand dynamic iteration stops the algorithm after executing 60 non-improving iterations. In order to evaluate their performances, the experiments are conducted for all job instances and the results are given in Figure 5.10. The average per cent deviations of 40-job is same for both criteria and a small difference is observed when 50-job and 100-job are considered. Dynamic stopping criterion performs better than fixed iteration for problem sets containing more than 40-jobs. Additionally, it shortens the execution time of the algorithm for 40-job and 50-job instances where the optimal solution can be found in very early stages of the search. For instance, after finding the global optimum solution at the iteration 20, algorithm with fixed stopping criteria repeats its search procedure 130 times without obtaining a better value. This is an undesirable and time- consuming operation. In contrast, dynamic stopping criterion allows the algorithm to implement 60 more iterations before terminating the search, which reduce the execution time. See Table 5.12.

Table 5.10. HCSS for SMTWT – comparison of alpha strategies

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	2.0>0.5+Δ	150
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	0.5+Δ	150
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	2.0>0.5	150

Table 5.11. HCSS for SMTWT – fine tuning for alpha increment

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	0.5+Δ	150

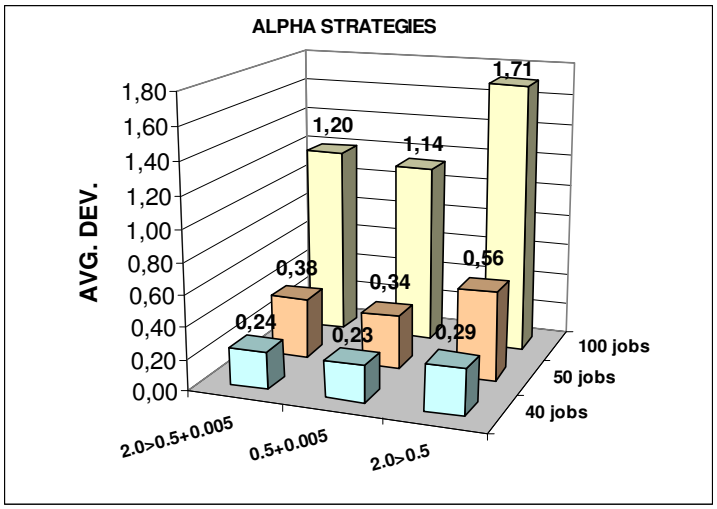

Figure 5.8. Experimental results for alpha strategies

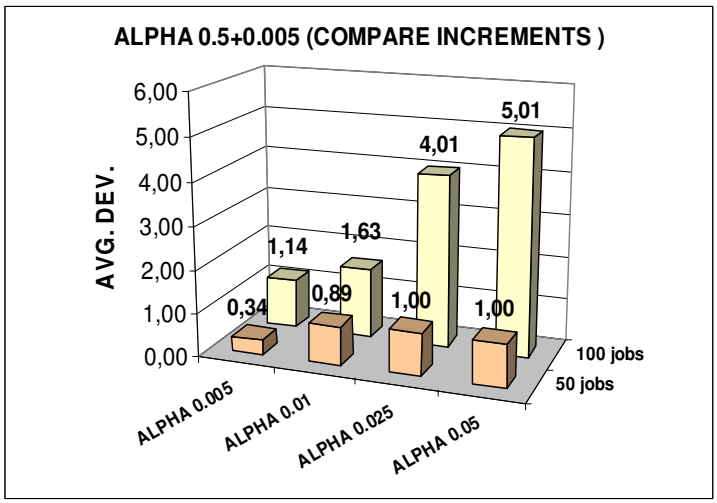

Figure 5.9. Fine-tuning of alpha increment for new alpha strategy

In the early development stages of our HCSS approach, 150 iteration is considered sufficiently long to conclude the search. Absence of necessary diversification tools and inaccurate organization of intensification procedures cause the meta-heuristic to get trapped at a local optimum at the very beginning. So, continuing the search procedure more than 150 iterations at this level seems useless. However, the presence of new dynamic alpha strategy cooperates with new MVNS methodology, prevents undesired early convergences to local optima and help to acquire better solutions towards to the end of the

Table 5.12. Experimental results for selected 40-job instances

		FIXED ITERATION			DYNAMIC ITERATION		
INSTANCE	OPTIMUM	Best Found	Iteration to Best	Total Iteration	Best Found	Iteration to Best	Total Iteration
001	913	913	2	150	913	2	62
002	1225	1225	11	150	1225	11	71
003	537	573	10	150	573	10	70
004	2094	2094	12	150	2094	12	72
005	990	990	11	150	990	11	71
006	6955	6955	8	150	6955	8	68
007	6324	6324	2	150	6324	2	62
008	6865	6865	10	150	6865	10	70
009	16225	16225	12	150	16225	12	72
010	9737	9737	89	150	9737	64	124
011	17465	17562	32	150	17465	71	131
012	19312	19312	57	150	19312	43	103
013	29256	29368	9	150	29368	9	69
014	14377	14432	3	150	14432	3	63
015	26914	26914	3	150	26914	3	63
016	72317	72317	142	150	72317	47	107
017	78623	78623	12	150	78623	12	72
018	74310	74387	43	150	74318	51	111
019	77122*	77432	16	150	77432	16	76
020	63229	63817	9	150	63817	9	69
021	77774	77774	17	150	77774	17	77
022	100484	100484	128	150	100484	65	125
023	135618	135618	2	150	135618	2	62
024	119947	119947	3	150	119947	3	63
025	128747	128747	4	150	128747	4	64

* Best Known

search. Table 5.13 presents the fitness values of best visited solutions and iteration elapsed up to best for selected 100-job instances. Based on results reported in the table, for some instances of fixed iteration stopping criterion, iteration number at which the best is found, is so close to constant stopping threshold. Therefore, the algorithm is terminated before investigating the promising neighborhoods of the current best solution. On the other hand, dynamic stopping allows it to execute at least 60 more iterations to explore the search space with new best solution. Although, the execution time is increased, better solutions can be visited by applying exploration or exploitation procedures within this extra time. Therefore, dynamic stopping criterion is adapted to for further experimentations.

After performing remarkable achievements with seeded initial solution set, a dynamic diversification strategy, a complete intensification method, a reliable dynamic

stopping procedure, we focus on the brain of our HCSS meta-heuristic; the reference set. So far, reference set is constructed by using 10 best solutions found so far (2-tier design) and 10 diverse solutions selected according to start or rank distance measurement method. At this stage, two arguments are tested to see whether they have a contribution to the performance of the reference set.

Table 5.13. Experimental results for selected 100-job instances

		FIXED ITERATION			DYNAMIC ITERATION		
INSTANCE	Best Known	Best Found	Iteration to Best	Total Iteration	Best Found	Iteration to Best	Total Iteration
010	53208	53992	142	150	53663	139	199
015	172995	178050	138	150	176539	154	214
019	477684	483201	147	150	480463	161	221
024	744287	744290	126	150	744290	120	180
033	32964	34593	140	150	33740	169	229
049	656693	656734	133	150	656715	172	232
057	11539	12250	118	150	12250	113	173
060	19912	19919	121	150	19919	108	168
064	100788	111047	139	150	104976	190	250
075	575274	575288	135	150	575288	131	191
086	66850	72942	129	150	72942	114	174
112	174367	177522	141	150	177522	148	208
113	91169	94436	123	150	94436	123	183
122	570459	570724	115	150	570724	108	168
123	397029	397848	137	150	397848	163	223
124	431115	431284	105	150	431284	93	153

The first argument is that a mixture of diverse solutions which are selected by rank and start distance measurement methods *(mix distance)* may provide different quality, dissimilar solution structures for the reference set and increase the diversity observed among the next generations. The experimentations with mix diverse solution selection is performed as described in Section 4.1.3 and the results for both mix distance and start distance selections are given in Figure 5.11. For the 40-job instances, mix distance with pool size 100 decreases the average per cent deviation to 0.14 which equals to 0.64 for start 100. Also for both 40-job and 50-job problem set, the lowest deviation value achieved so far is obtained by using mix diverse solution selection with pool size 85; 0.11 and 0.31 respectively.

Table 5.14. HCSS for SMTWT – comparison of stopping criteria

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	0.5+Δ	150
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC

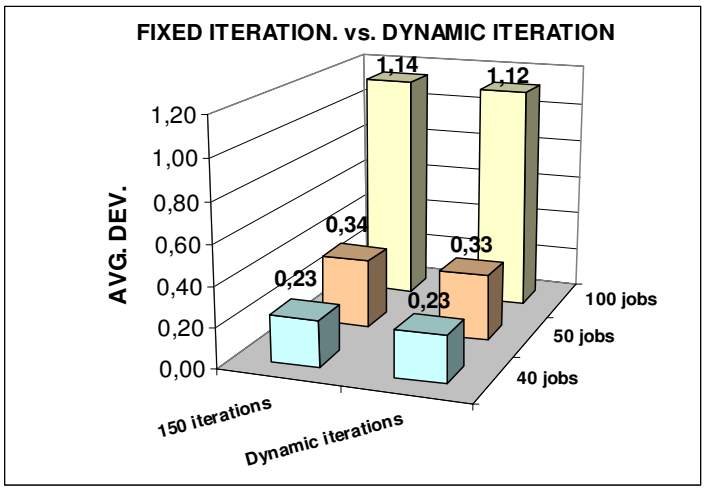

Figure 5.10. Effect of dynamic stopping criteria

The second argument is about the reference set update method. The new update method as mentioned in Section 4.1.3, is aiming to preserve diversity continuously, instead of allowing it to become homogenous by only admitting solutions from one source, which tend to have very similar components at further stages of the search (3-tier design). A reference set whose best solutions are updated from two sources namely the reserve list and the pool, may lead the search at a better direction. After conducting necessary experiments to evaluate the second argument, obtained results are reported in Figure 5.12. According to the results, selecting diverse solutions using both start and rank distance measurement method reduces the average per cent deviation value to 0.11 for 40-job and 0.31 for 50-job instances, which are initially 0.23 and 0.33 for start diverse solutions. However, new 3-tier reference set update method performs better than the diverse solution selection method and comes up with a deviation 0.07 and 0.012 for 40-job and 50-job problems respectively. Finally, mix diverse solution selection procedure and 3-tier reference set design are employed together to test their cooperative performance. Heterogeneous transient best solutions with dissimilarly evaluated diverse solutions

Table 5.15. HCSS for SMTWT – evaluation of mix diverse solution selection method

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
SEED-SCR	2-TIER	START	100, 85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC
SEED-SCR	2-TIER	MIX	100,85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC

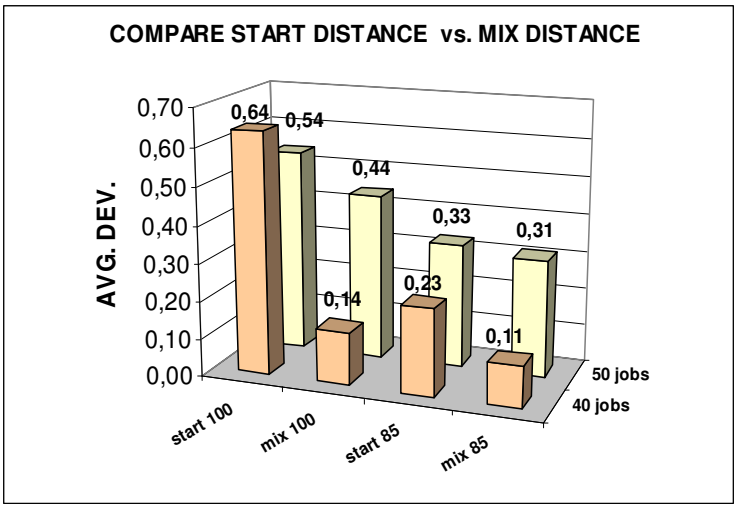

Figure 5.11. Comparison of mix diverse and start diverse solution selection methods

provide a more extensive exploration through the promising regions; thus increase the chance of visiting better optimums. This argument can easily be proven by investigating the experimental results for 40-job and 50-job instances. Both per cent deviation values decrease to 0.04 and 0.09 respectively. As a result, new diverse solution selection method with new reference set design is adapted to our HCSS approach and this concludes the experimentation phase conducted to developed the algorithm in a sequential manner.

Table 5.16. HCSS for SMTWT – 3-tier design of reference set

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
SEED-SCR	2-TIER	START	85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC
SEED-SCR	2-TIER	MIX	85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC
SEED-SCR	3-TIER	START	85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC
SEED-SCR	3-TIER	MIX	85	MVNS ALL	4-5-6	0.5+Δ	DYNAMIC

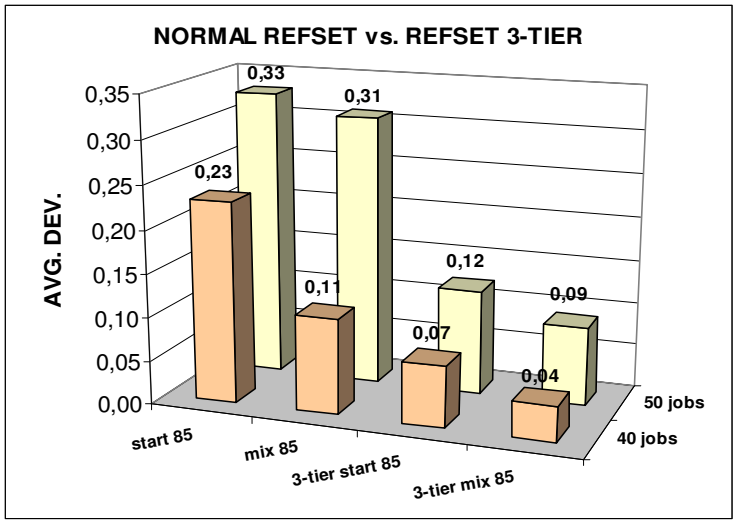

Figure 5.12. Effects of new reference set update methods

5.4.1. Final Results

After conducting necessary numerical experimentations to adapt new improvement methods for our HCSS approach an ultimately modified algorithm, which can be implemented on all job sets, is achieved. Different from the elementary model, the fine-tuned algorithm composed of a new initial solution generation method (SEED-SCR), an advance reference set update method (3-tier + mix distance), an extended intensification strategy (MVNS ALL) and a self-adjustable alpha parameter (0.5+Δ) which control the diversification of generated new solutions. In addition to that the dynamic stopping criterion allows the algorithm to decide its own termination time depending on the search performance. This final form of the HCSS approach is now tested with 40-job, 50-job and 100-job problem sets respectively and the obtained results are presented.

Due to the probabilistic nature of solution combination method, five different seeds are used for the random number generator employed in the HCSS algorithm. The seeds are selected to be 1,3,7,9 and 24. The performance measures is again average per cent deviation which is calculated as described previously and computational times are reported as Average CPU (Entire Search) and Average CPU (Time elapsed up to the best). The first measurement indicates the total time allowed for HCSS before termination and the second one is the amount of time elapsed in the search until the best solution for that instance is

found. The deviation and computational time results presented in this section show the average of all final outcomes which are obtained with the selected seeds separately. The minimum and maximum values of these results for each instance are given in Appendix F Table F.5 to Table F.7.

Table 5.17. Final results for HCSS approach – 40-job problem set- SMTWT

40-JOB PROBLEM SET				
METHOD		**SS**	**SS + FVNS**	**HCSS**
REFERENCE SET	***Best***	10	10	10
	Diverse	10	10	10
	Method	3-tier + mix	3-tier + mix	3-tier + mix
INITIAL SOLUTION SET	***Size***	150	150	150
	Method	SEED-SCR	SEED-SCR	SEED-SCR
POOL	***Size***	85	85	85
SUBSET GENERATION	***Type***	size-2	size-2	size-2
SOLUTION COMBINATION METHOD	***Type***	BLX-α	BLX-α	BLX-α
	α	0.5	0.5	0.5
ALPHA STRATEGY	***Type***	Dynamic	Dynamic	Dynamic
	When	after 30	after 30	after MVNS
	α	0.5	0.5	0.5
	Δ	0.005	0.005	0.005
INTENSIFICATION STRATEGY	***Type***	NONE	VNS	VNS
	Type of VNS	NONE	FINAL	MIDDLE
	Search depth	NONE	4-5-6	4-5-6
	When	NONE	after SS	after 30
STOPPING CRITERION	***Type***	Dynamic	Dynamic	Dynamic
	When	after 60	after 60	after 60
AVERAGE PER CENT DEV (5 seed averaged)		0.20	0.15	0.03
AVERAGE PER CENT DEV (minimum)		0.17	0.10	0.01
AVERAGE CPU (Entire search-5 seed averaged)		37.80	37.97	71.19
AVERAGE CPU (Time elapsed up to best 5 seed averaged)		7.81	24.36	16.93

In order to illustrate the performance of our hybrid method (HCSS), it is compared against two SS approaches; one with no intensification strategy and one with a final VNS. SS algorithms used in comparison contain same components and same parameter values as the ones utilized by HCSS. The only difference is the absence of an intensification strategy. The structure of compared algorithms and employed advanced methods are all given in Table 5.17 to the Table 5.19. All algorithms are executed until a predefined stopping criterion and come up with a best solution for each instance. For the final VNS

case, the best solution found by the SS is explored intensively by using a VNS as described in Section 4.1.6. The execution times of all methods reported in tables are all given in terms of non-improved iterations, i.e. the term after 30 cites that the method is implemented after 30 non-improved consecutive iterations.

Table 5.18. Final results for HCSS approach – 50-job problem set-SMTWT

50-JOB PROBLEM SET				
METHOD		**SS**	**SS + FVNS**	**HCSS**
REFERENCE SET	***Best***	10	10	10
	Diverse	10	10	10
	Method	3-tier + mix	3-tier + mix	3-tier + mix
INITIAL SOLUTION SET	***Size***	150	150	150
	Method	SEED-SCR	SEED-SCR	SEED-SCR
POOL	***Size***	85	85	85
SUBSET GENERATION	***Type***	size-2	size-2	size-2
SOLUTION COMBINATION METHOD	***Type***	BLX-α	BLX-α	BLX-α
	α	0.5	0.5	0.5
ALPHA STRATEGY	***Type***	Dynamic	Dynamic	Dynamic
	When	after 30	after 30	after MVNS
	α	0.5	0.5	0.5
	Δ	0.005	0.005	0.005
INTENSIFICATION STRATEGY	***Type***	NONE	VNS	VNS
	Type of VNS	NONE	FINAL	MIDDLE
	Search depth	NONE	4-5-6	4-5-6
	When	NONE	after SS	after 30
STOPPING CRITERION	***Type***	Dynamic	Dynamic	Dynamic
	When	after 60	after 60	after 60
AVERAGE PER CENT DEV (5 seed averaged)		0.51	0.38	0.11
AVERAGE PER CENT DEV (minimum)		0.47	0.31	0.09
AVERAGE CPU (Entire search-5 seed averaged)		55.96	57.14	95.47
AVERAGE CPU (Time elapsed up to best 5 seed averaged)		20.98	44.77	31.23

According to the results given in Table 5.17 to Table 5.19, HCSS algorithm clearly outperforms the other algorithms. By applying an intensification strategy, the deviation value 0.20 is decreased to 0.03 for 40-job problem set, from 0.51 to 0.11 for 50-job and from 1.07 to 0.50 for 100-job problem set. Although the execution times of HCSS are longer than normal SS, this handicap seems negligible when the reduction in average per cent deviation is considered with respect to this extra time. Therefore our intensification

strategy proves its quality and performance when integrated with the Scatter Search methodology.

Table 5.19. Final results for HCSS approach – 100-job problem set-SMTWT

100-JOB PROBLEM SET				
METHOD		**SS**	**SS + FVNS**	**HCSS**
REFERENCE SET	***Best***	10	10	10
	Diverse	10	10	10
	Method	3-tier + mix	3-tier + mix	3-tier + mix
INITIAL SOLUTION SET	***Size***	150	150	150
	Method	SEED-SCR	SEED-SCR	SEED-SCR
POOL	***Size***	85	85	85
SUBSET GENERATION	***Type***	size-2	size-2	size-2
SOLUTION COMBINATION METHOD	***Type***	BLX-α	BLX-α	BLX-α
	α	0.5	0.5	0.5
ALPHA STRATEGY	***Type***	Dynamic	Dynamic	Dynamic
	When	after 30	after 30	after MVNS
	α	0.5	0.5	0.5
	Δ	0.005	0.005	0.005
INTENSIFICATION STRATEGY	***Type***	NONE	VNS	VNS
	Type of VNS	NONE	FINAL	MIDDLE
	Search depth	NONE	4-5-6	4-5-6
	When	NONE	after SS	after 30
STOPPING CRITERION	***Type***	Dynamic	Dynamic	Dynamic
	When	after 60	after 60	after 60
AVERAGE PER CENT DEV (5 seed averaged)		1.07	0.65	0.50
AVERAGE PER CENT DEV (minimum)		0.98	0.53	0.44
AVERAGE CPU (Entire search-5 seed averaged)		138.33	145.02	236.69
AVERAGE CPU (Time elapsed up to best 5 seed averaged)		70.95	140.85	144.18

Comparing the performance of HCSS with respect to given problem sets, it is obvious that our algorithm is significantly sensitive to the size of the problem set. For 40-job and 50-job instances it performs satisfactorily and solves optimally 88.64 per cent and 76.64 per cent of the 125 instances. If the best performing seed is considered, then these percentages become 90.6 for 40-jobs and 78.4 for 50-job problem set. On the other hand, best known solutions found for 100-job problem set drop below 50 per cent of the total instances. This is simply due to the fact that as the problem size increases, the problem becomes harder to solve. This hardness is reflected on the algorithm as “difficulty to

converge" to the global optimum. Although, the performance of HCSS is not so adequate when finding the best known solutions, the algorithm succeeds to solve this harder problem set with a 0.50 average per cent deviation per instance.

To sum up, the scope of this study is to develop a meta-heuristic approach, which is robust to all problem sets. Instead of constructing problem-specific algorithms which are using different parameter values depending on the problem size, a unique algorithm is established aiming to come up with minimum average per cent deviation for all problem sets. In other words, HCSS approach developed in this section utilized same components and same parameter values for each problem set. During the numerical experimentation, the overall performance is taken into account when adapting advanced methods to the elementary HCSS model. In addition to that better results for some problem instances were found by tested strategies which is different than the adapted one, but they were not inserted to the main algorithm because of their problem-specific achievements. The best found solutions during the numerical experimentation are given in Appendix F.

5.5. Numerical Experimentation for PMTT

The basic model employed for the preliminary experimentation of the PMTT problem, uses a reference set consists of 10 high quality and 10 diverse solutions. The initial solutions are generated by EDD method which is described in Section 4.2.2. Initial solution set size is considered as 150. Best solutions are selected from the reserve list whereas diverse solutions are chosen according to their start time distances to the reference set. The solution combination method employs BLX-α operator where α equals to 0.5 and the generated trial solutions are stored in the pool which has a room for 100 offspring. In this elementary model, no intensification strategy such as MVNS or FVNS is performed. The dynamic stopping rule as aforementioned performs at least as good as fix iteration termination rule and in addition to that, it prevents the algorithm to end before investigating lately found best solutions' neighborhoods which may be observed in fix iteration situation. Therefore, the stopping criterion in the basic algorithm is decided to be the dynamic rule as implemented in SMTWT problem. The used components and fix parameter values of our basic HCSS algorithm is presented in Table 5.20.

Table 5.20. Basic HCSS algorithm- PMTT

	BASIC HCSS
REFERENCE SET	
UPDATE METHOD	2-tier + start
# of BEST	10
# of DIVERSE	10
INITIAL SOLUTION SET	
GENERATION METHOD	EDD
# of INITIAL SOL's	150
POOL	
SIZE	100
SOLUTION COMBINATION METHOD	
ALPHA STRATEGY	STATIC
ALPHA VALUE	0.5
INTENSIFICATION STRATEGY	
TYPE OF VNS	NO
SEARCH DEPTH	X
STOPPING CRITERION	
TYPE	DYNAMIC
# of NON-IMPROVING ITERATIONS	50

As the first step of experimentation, two diverse selection methods *(rank and start)*, two initial solution generation methods *(EFT and EDD)* and three pool sizes *(ALL , 100 and 85)* are tested by taking all their combinations (See Table 5.21) . The aim is to eliminate those strategies that do not yield satisfactory results and to retain the most promising combinations to further analyses. Figure 5.13 presents the results for this step. From the figure, it can be seen that all combinations containing enlarged pool size (ALL) result in poor optimums when compared to restricted pool sizes. Hence, the pool including all generated subsets will not be utilized for the following experimentations. Pool size 100 seems outperforming beyond the other pool sizes; but more experimentation with different combinations at further steps of the development procedure should be performed to prove its superiority. When comparing initial solutions sets, it is obvious that the final results are highly unsteady depending on the elements of HCSS algorithm, to which initial solutions generation method combines. For instance, it is not a consistent conclusion that EFT is the best performing initial solutions generation method by looking at a few exceptions. Although the best average per cent deviation value is obtained by *EFT rank 100* as 6.68, *EFT start ALL* and *EFT start 85* result in 16.89 and 10.85 respectively which are the worst deviations observed according to pool size categorization. Same situation is noted once

evaluating the diverse solutions selection methods. EFT rank 100 grants best average per cent deviations whereas *EDD rank 100* comes up with a value 13.58. Thus, no exact choice between start and rank diverse solutions selection methods can be made at this level. Since the pool size of our HCSS can not be determined, the experimental results related to tested combinations are divided into two groups according to their pool sizes for further analyses. Experimental data corresponding to pool size 100 and pool size 85 will be given in separate figures different than each other.

Table 5.21. HCSS for PMTT - comparison of pool sizes and distance measurement methods

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
EFT	2-TIER	START	ALL,100,85	NO	X	0.5	DYNAMIC
EDD	2-TIER	RANK	ALL,100,85	NO	X	0.5	DYNAMIC
EFT	2-TIER	RANK	ALL,100,85	NO	X	0.5	DYNAMIC
EDD	2-TIER	START	ALL,100,85	NO	X	0.5	DYNAMIC

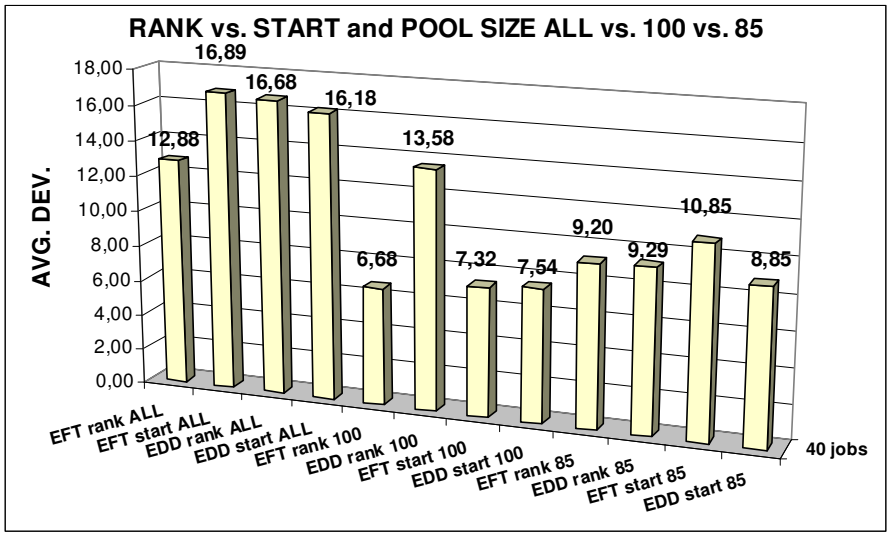

Figure 5.13. Comparison results for pool sizes and distance measurement methods

The next step is to determine a suitable VNS depth for our HCSS approach. During the experimentation phase, middle VNS (*type #2*) is employed as described in section 4.2.4. Again, MVNS is triggered at the end of consecutive 30 non-improved iterations. Since PMTT problem consist of both sequencing and allocation problem, we neglected large search depths, which cause heavy computational burdens for such a challenging

problem. In contrast, small depth values can not provide an efficient neighborhood generation tool and due to its narrow-scope exploration, the intensification strategy becomes a superficial method. For this reason, search depth 3-4-5 and 4-5-6 are considered as the most suitable variable depths according to their performance-time ratio and tested with MVNS. The results are given in Figure 5.14 and Figure 5.15. Examining Figure 5.14, the performances of MVNS 3-4-5 and MVNS 4-5-6 are very close to level of being almost identical. MVNS 4-5-6 outperforms its candidate slightly among three of the four tested combinations whereas *EFT start 85* cooperated with MVNS 3-4-5 shows an exceptional success. The required proof to conclude our comparison comes from the experiments conducted with pool size 100. Although the overall average per cent deviation values are not as good as we found with pools size 85, the performance difference can be clearly observed among all columns of the Figure 5.15. Since, we are not concerned with the best combination of pool sizes, initial sets or diverse solution methods; aiming only to investigate the search depth at this stage; 4-5-6 seems an appropriate choice as the depth for variable neighborhood search.

Table 5.22. HCSS for PMTT - comparison of search depths

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
EFT, EDD	2-TIER	START	100,85	MVNS	3-4-5	0.5	DYNAMIC
EFT, EDD	2-TIER	RANK	100,85	MVNS	4-5-6	0.5	DYNAMIC
EFT, EDD	2-TIER	RANK	100,85	MVNS	3-4-5	0.5	DYNAMIC
EFT, EDD	2-TIER	START	100,85	MVNS	4-5-6	0.5	DYNAMIC

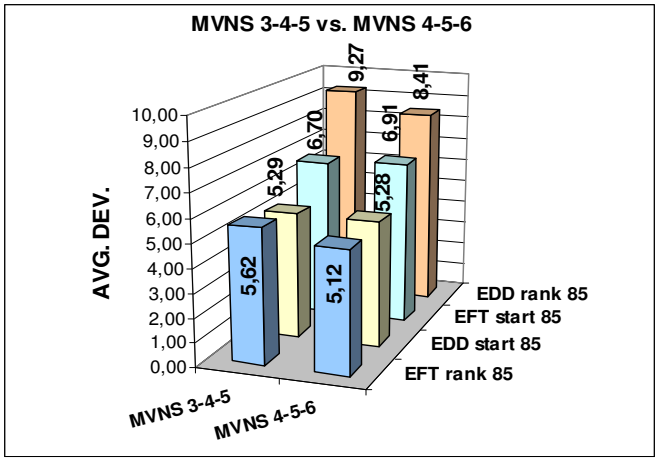

Figure 5.14. Experimental results for compared search depths (pool size 85)

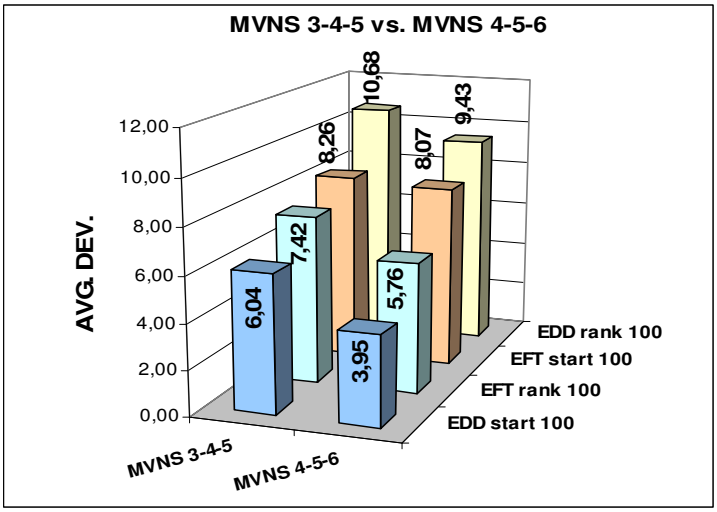

Figure 5.15. Experimental results for compared search depths (pool size 100)

Despite the fact that the best performed search depth is selected, it seems a bit time consuming when employed with MVNS, Therefore, the final VNS is revisited as an intensification alternative which may reduce the execution time without altering the overall performance of the neighborhood search. The experimental results for MVNS 4-5-6 and FVNS 4-5-6 are reported in Figure 5.16 and Figure 5.17 together with the results achieved when no intensification strategy is used. According to the values displayed in both of the figures, MVNS 4-5-6 outperforms FVNS and no VNS noticeably. A final neighborhood search shortens the time needed to implement the algorithm as expected, but it definitely diminishes the performance of intensification strategy. Thus, MVNS 4-5-6 is again adopted for HCSS approach.

Table 5.23. HCSS for PMTT - comparison of different intensification strategies

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
EFT, EDD	2-TIER	START	100,85	MVNS	4-5-6	0.5	DYNAMIC
EFT, EDD	2-TIER	RANK	100,85	MVNS	4-5-6	0.5	DYNAMIC
EFT, EDD	2-TIER	START	100,85	FVNS	4-5-6	0.5	DYNAMIC
EFT, EDD	2-TIER	RANK	100,85	FVNS	4-5-6	0.5	DYNAMIC
EFT, EDD	2-TIER	START	100,85	NO	X	0.5	DYNAMIC
EFT, EDD	2-TIER	RANK	100,85	NO	X	0.5	DYNAMIC

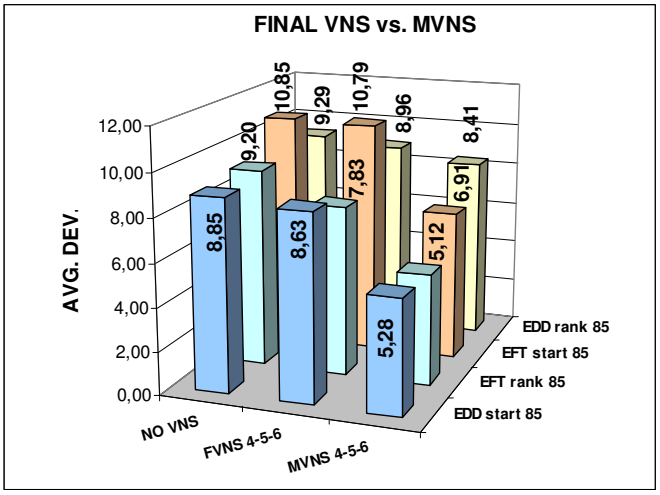

Figure 5.16. Effect of intensification strategy for PMTT problem (pool size 85)

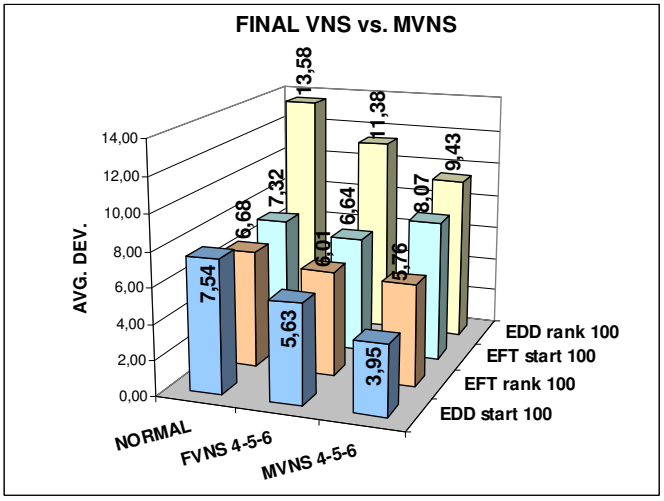

Figure 5.17. Effect of intensification strategy for PMTT problem (pool size 100)

As we mentioned earlier, initial solution generations method is one of the most crucial component of SS. A simple and an applicable prescription for a well-constructed initial solution set is to choose diverse high quality solutions, which are situated in far regions of solutions space and to seed this distinctly structured initial solution set with good solutions provided by list heuristics. EFT and EDD are two methods that are used initially as depicted in Section 4.2.2 EFT method generate somewhat more structured results than EDD method. It strictly depends on the ready time of the jobs. Start time of each job is selected randomly from an interval generated by the formula 4.6. Then these

start times are sorted in order of non-decreasing times and a sequence is obtained. Two initial trial solutions with similar sequences mostly result in close or even same total tardiness values. For the problem instances where given ready times are not so adjacent, the intervals which are formed with respect to these ready times, do not interfere with each other frequently. As a result, randomly selected start times can not provide sufficient alternatives when sorted and the sequence is somehow similar to the one obtained after sorting jobs according to their non-decreasing ready times. In this manner, initial solution set occupies the part of the search space dominated by solutions generated by earliest ready time dispatching rule. Hence, it is a valid argument that this set is a collection of similar solutions and limits the diversity of initial reference set.

Contrarily, consider EDD method where the jobs are assigned to the machines randomly and then sorted according to their due dates. Job with an earlier completion necessity gets the precedence to be processed on that assigned machine. This type of initial solution method meets the demand for required diverse solutions but fails to generate elite solutions of adequate quality. The reason behind this handicap is that EDD dispatching rule is mainly developed for single machine scheduling problem and stand alone can not respond parallel machine problems efficiently. In order to compensate inability of EDD method while creating good solutions, we developed an advanced version called multi-rule method. In addition to EDD rule, SPT and XR&M rules are also utilized to order jobs on a given machine as explained in Section 4.2.2. By doing so, all alternative dispatch orders resulted from employed heuristics are compared and the best among all is selected. Consequently, it is guaranteed that even in the worst case, the initial solution will be as good as the solutions obtained by EDD rule. This advance version therefore improves the solution quality related to total tardiness value.

To demonstrate structural characteristics and variances between the individuals in the initial solution set, a randomly selected problem instance is taken into account and three different sets are constructed by using predefined generation methods. The results are given in Figure 5.18. The best known total tardiness value found in the literature is considered as the lower bound and the deviations are calculated as follows;

$$\text{deviation} = (\text{initial solutions' fitness value} - \text{lower bound}) / \text{lower bound} \quad (5.5)$$

According to these results, EFT shows a uniform distribution, whereas EDD and multi-rule provide non-homogenous chaotic distributions. This chaotic distribution indicates the dissimilarity of the internal structure and grant different quality diverse solutions. In addition to that, multi-rule behaves as a regulation mechanism that adjusts dispatching rules, screens the solutions with same fitness value and improve the overall quality of the initial pool. Therefore it can be called as structured family of a chaotic population.

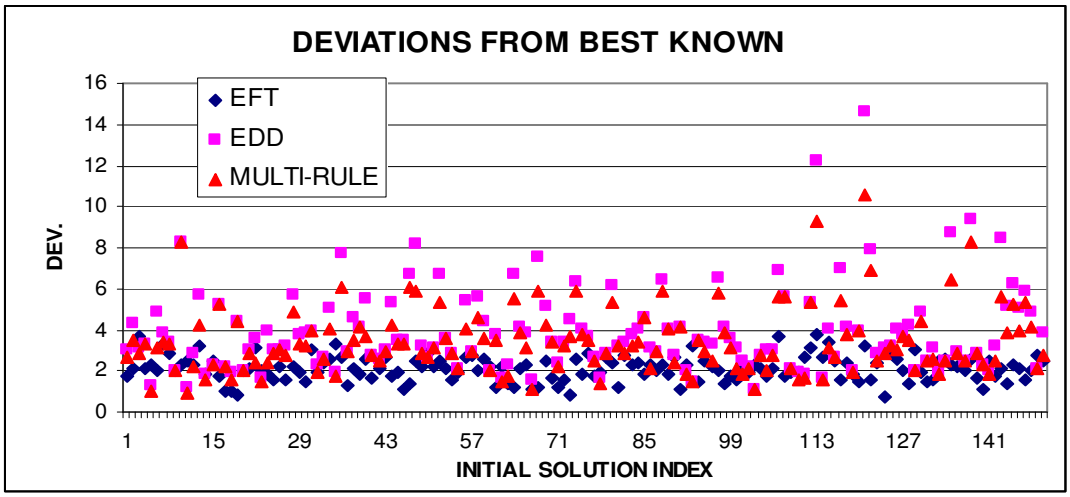

Figure 5.18. Structural characteristics of initial solution set for three different methods

The experimentations are conducted to compare these three initial solutions generation methods. Instead of considering all diverse solution selection methods and pool size combinations, only the best performing combinations; namely start 100, start 85 and rank 85 are tested with initial solution method. The results are shown in Figure 5.20. It is obvious that multi-rule is superior when compared to the previous methods in all cases and reduces the best average per cent deviations of 3.95 to 3.11. So, multi-rule strategy is included in our HCSS approach.

Table 5.24. HCSS for PMTT - comparison of different initial solution generation methods

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
EFT, EDD	2-TIER	START	100,85	MVNS	4-5-6	0.5	DYNAMIC
EFT, EDD	2-TIER	RANK	85	MVNS	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	START	100,85	MVNS	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	RANK	85	MVNS	4-5-6	0.5	DYNAMIC

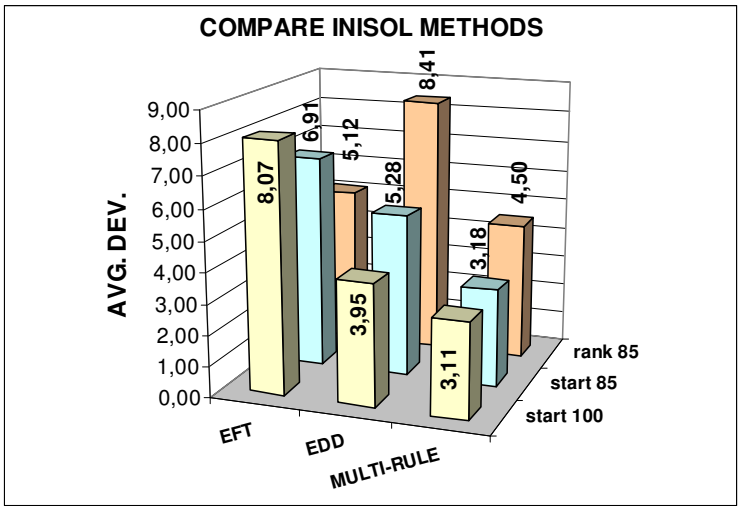

Figure 5.19. Experimental results for initial solution generation methods for PMTT

In order to implement a comprehensive intensification procedure, VNS should generate all possible neighborhoods based on both machine and job sequences. As described in Section 4.2.4, three different types of VNS approaches are developed and applied in succession to all elite elements of reference set (MULTISTEP). This intensification strategy aims to accomplish two main goals. First goal is to explore all reachable neighborhoods which can be originated from a single solution and the second one is to examine all high quality solutions which may promise better results.

In the numerical analyses, the new strategy is combined with the new initial solution generation method and the results are compared with previous intensification method that utilizes single step MVNS to first three elite solutions. The first type VNS is triggered at the end of 15 consecutive non-improved iterations, second type is implemented after 25 non-improved iterations and finally last type is executed when the best solution VNS counter indicates the number 35. During the preliminary experimentations of the multi-step VNS, it is discovered that the first type VNS with search depth 4-5-6 causes a great computational burden to the algorithm because of the corresponding outnumbered machine assignment alternatives. Therefore, the search depth for the first type VNS is decreased to 3-4-5 as an exception in order to reduce intensification strategy execution time. This exception is only valid for the first type, whereas the second and third type VNS techniques are implemented with 4-5-6. The average per cent deviations are given in

Figure 5.20. As expected, new intensification strategy outperforms its superficial candidate and reduces deviations from 3.11 to 1.71 for start 100, from 3.18 to 1.98 for start 85 and from 4.50 to 2.96 for rank 85. Hence, the new strategy is adapted to HCSS approach for PMTT.

Table 5.25. HCSS for PMTT - comparison of new intensification strategy

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
MULTI	2-TIER	START	100,85	MVNS	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	START	100,85	MVNS ALL	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	RANK	85	MVNS	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	RANK	85	MVNS ALL	4-5-6	0.5	DYNAMIC

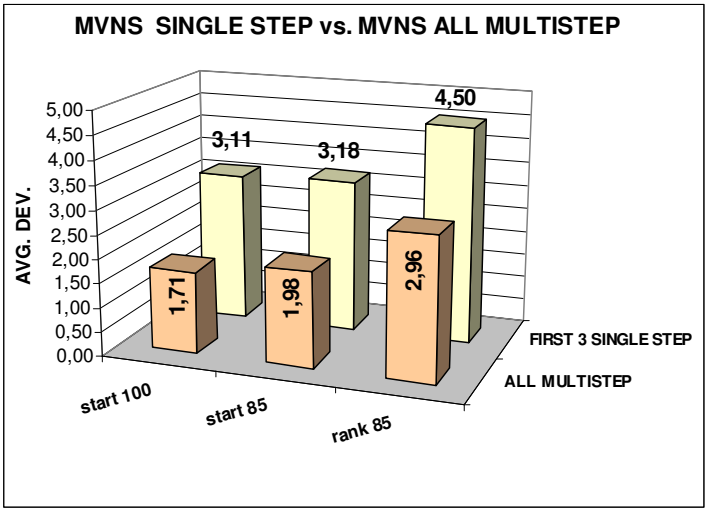

Figure 5.20. Effect of new intensification strategy

Another important issue is to investigate consistency of the dynamic alpha strategy for PMTT problem. In this strategy, α is fixed at 0.5 during a period of time determined by a switch. When the switch becomes on, α is increased by 0.005 at the end of every iteration that passes without achieving any improvement. Mentioned strategy is tested throughout PMTT problem instances by conducting experiment that utilize HCSS approach stated in Table 5.26. The corresponding results are presented in Figure 5.21. The superiority of dynamic alpha strategy can be proven by looking at the deviation reductions among all cases, where the best found average per cent deviation value is decreased to 1.56 for start 100 combination.

Table 5.26. HCSS for PMTT – comparison of alpha strategies

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
MULTI	2-TIER	START	100,85	MVNS ALL	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	RANK	85	MVNS ALL	4-5-6	0.5	DYNAMIC
MULTI	2-TIER	START	100,85	MVNS ALL	4-5-6	0.5+	DYNAMIC
MULTI	2-TIER	RANK	85	MVNS ALL	4-5-6	0.5+	DYNAMIC

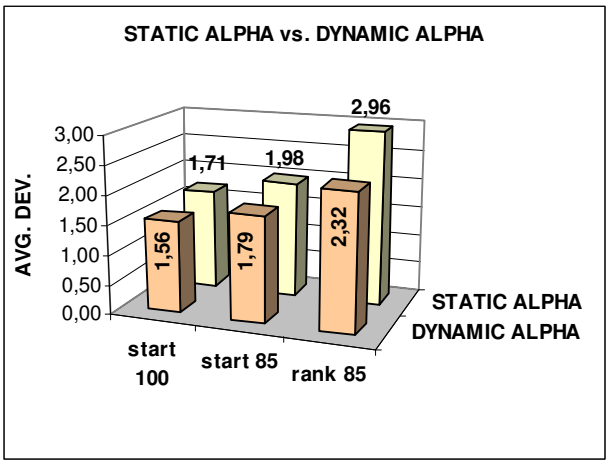

Figure 5.21. Comparison of static alpha vs. dynamic alpha

Finally, the reference set update method is revisited and preceding improvement techniques, which are used in SMTWT, are experimented for parallel machine scheduling problem. Starting with the diverse solution selection method, a mixed selection procedure is implemented, where half of diverse solutions are selected by start distance and other half is selected using rank distance measurement method. According to the results reported in Figure 5.22 mixed selection with pool size 100 shows a better performance with respect to start distance with same pool size. On the other hand, mix 85 comes up with an average per cent deviation 1.81 which stays between 1.79 of start 85 and 2.32 of rank 85. Hence the three combinations with the lowest deviations (start 100, mixed 100 and mixed 85) are promoted to the final experimentation which will be conducted to verify the success of 3-tier design for PMTT problem.

Table 5.27. HCSS for PMTT – comparison of mix diverse solution selection method

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
MULTI	2-TIER	START	100,85	MVNS ALL	4-5-6	0.5+	DYNAMIC
MULTI	2-TIER	RANK	85	MVNS ALL	4-5-6	0.5+	DYNAMIC
MULTI	2-TIER	MIX	100,85	MVNS ALL	4-5-6	0.5+	DYNAMIC

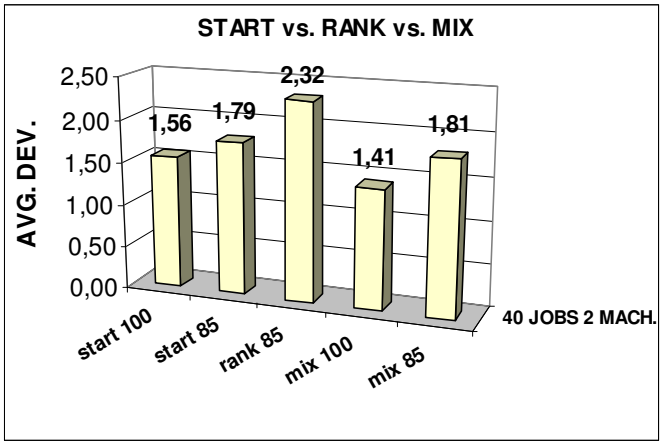

Figure 5.22. Numerical analysis for diverse solution selection methods

Table 5.28. HCSS for PMTT – 3-tier design of reference set

INISOL	REFSET	DISTANCE	POOL	VNS	DEPTH	ALPHA	STOP
MULTI	2-TIER	START	100	MVNS ALL	4-5-6	0.5+	DYNAMIC
MULTI	3-TIER	MIX	100,85	MVNS ALL	4-5-6	0.5+	DYNAMIC

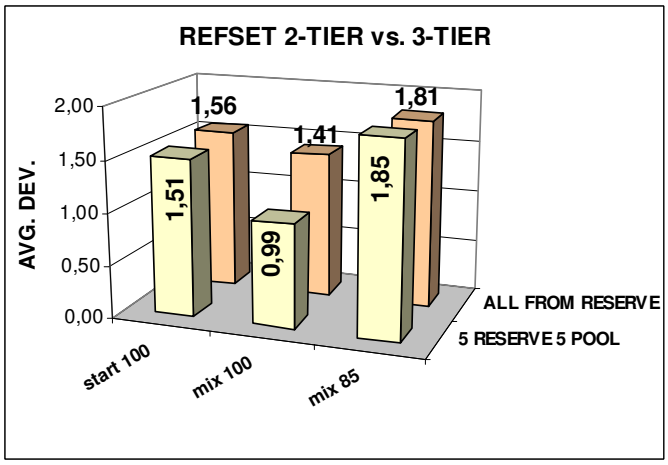

Figure 5.23. Effects of new reference set update methods for PMTT

Due to its elite solution selection procedure, 3-tier design provides fresh solutions for the reference set at the end of each iteration. These newly visited solutions increase the chance of obtaining a better optimum during neighborhood search which is performed for all elites. Based on this argument, new reference set update method is employed for HCSS

approach as the final improvement strategy that concludes experimental analysis section. Being modified with the earlier developments which are presented in Table 5.28, HCSS algorithm is executed for numerical analysis and the results are reported in Figure 5.23. The average per cent deviation at the final stage is now reduced to 0.99 which is accomplished by the cooperation of new diverse selection method and new reference set design. Therefore, considered reference set update method is adapted to HCSS approach and the experimental analysis section is ended.

5.5.1. Final Results

Depending on the nature of the PMTT problem, the improvement methods used in SMTWT problem can not be copied to our HCSS algorithm. Some of these methods are revised partially or constructed from the beginning in order to be employed in the algorithm. Then the aforementioned methods are tested and adapted to HCSS according to the comparative results obtained by numerical experimentations. At the end of analysis, the modified HCSS approach, which can be implemented for all job set problem, is achieved. Different from the ultimate model used in SMTWT, the fine-tuned algorithm composed of a new initial solution generation method that employs a multi-rule to generate initial set (MULTI), an advance reference set update method which contains different quality best and diverse solutions (3-tier + mix distance), an extended multi-step intensification strategy where three different types of VNS is implemented to explore all possible neighborhoods of the current solution (MVNS ALL) and a self-adjustable alpha parameter ($0.5+\Delta$) which control the diversification of generated new solutions. In addition to that the dynamic stopping criterion allows the algorithm to decide its own termination time depending on the search performance. This final form of the HCSS approach is now tested on four types of problems which are encountered with two main job sets: one with 40-jobs with 2 or 4 machines in parallel, the second with 60-jobs with 2 or 4 machines in parallel.

Since our HCSS approach for PMTT contains stochastic parameters, five different seeds are again used for the random number generator employed in the algorithm. The seeds are selected to be 1,3,7,9 and 24. The performance measures is again average per cent deviation which is calculated as described previously and computational times are reported as Average CPU (Entire Search) and Average CPU (Time elapsed up to the best).

The deviation and computational time results presented in this section show the average of all final outcomes which are obtained with the selected seeds separately. The maximum and minimum results obtained by different seeds are given in Table F.10 and Table F.11.

Table 5.29. Final results for HCSS approach – 40-job PMTT problem set

40-JOB PROBLEM SET					
METHOD		**2-MACHINE**		**4-MACHINE**	
		SS	**HCSS**	**SS**	**HCSS**
REFERENCE SET	***Best***	10	10	10	10
	Diverse	10	10	10	10
	Method	3-tier + mix	3-tier + mix	3-tier + mix	3-tier + mix
INITIAL SOLUTION SET	***Size***	150	150	150	150
	Method	MULTI	MULTI	MULTI	MULTI
POOL	***Size***	100	100	100	100
SUBSET GENERATION	***Type***	size-2	size-2	size-2	size-2
SOLUTION COMBINATION METHOD	***Type***	BLX-α	BLX-α	BLX-α	BLX-α
	α	0.5	0.5	0.5	0.5
ALPHA STRATEGY	***Type***	Dynamic	Dynamic	Dynamic	Dynamic
	When	25	after MVNS	25	after MVNS
	α	0.5	0.5	0.5	0.5
	Δ	0.005	0.005	0.005	0.005
INTENSIFICATION STRATEGY	***Type***	NONE	MVNS ALL	NONE	MVNS ALL
	Type of VNS	NONE	Multi-step	NONE	Multi-step
	Search depth	NONE	4-5-6	NONE	4-5-6
	When	NONE	after 15-25-35	NONE	after 15-25-35
STOPPING CRITERION	***Type***	Dynamic	Dynamic	Dynamic	Dynamic
	When	after 50	after 50	after 50	after 50
AVERAGE PER CENT DEV (5 seed averaged)		10.28	0.76	23.03	5.33
AVERAGE PER CENT DEV (minimum)		8.52	0.43	12.91	4.70
AVERAGE CPU (Entire search-5 seed averaged)		72.17	140.6	64.24	88.51
AVERAGE CPU (Time elapsed up to best 5 seed averaged)		24.72	102.17	51.38	70.81

Similar to the comparison performed in Section 5.4.1, HCSS approach is evaluated against the SS with no intensification strategy. SS algorithm employed in comparison contains same components and same parameter values used in HCSS, -except the intensification method. The main objective is to criticize the performance of hybrid approach with respect to the time penalty associated with computational burden of exploitation step. The structure of compared algorithms and employed advanced methods

are all given in Table 5.29 and Table 5.30. All algorithms are executed until a predefined stopping criterion and come up with a best solution for each instance. The execution times of all methods reported in tables are all given in terms of non-improved iterations.

Table 5.30. Final results for HCSS approach – 60-job PMTT problem set

60-JOB PROBLEM SET					
METHOD		**2-MACHINE**		**4-MACHINE**	
		SS	**HCSS**	**SS**	**HCSS**
REFERENCE SET	***Best***	10	10	10	10
	Diverse	10	10	10	10
	Method	3-tier + mix	3-tier + mix	3-tier + mix	3-tier + mix
INITIAL SOLUTION SET	***Size***	150	150	150	150
	Method	MULTI	MULTI	MULTI	MULTI
POOL	***Size***	100	100	100	100
SUBSET GENERATION	***Type***	size-2	size-2	size-2	size-2
SOLUTION COMBINATION METHOD	***Type***	BLX-α	BLX-α	BLX-α	BLX-α
	α	0.5	0.5	0.5	0.5
ALPHA STRATEGY	***Type***	Dynamic	Dynamic	Dynamic	Dynamic
	When	25	after MVNS	25	after MVNS
	α	0.5	0.5	0.5	0.5
	Δ	0.005	0.005	0.005	0.005
INTENSIFICATION STRATEGY	***Type***	NONE	MVNS ALL	NONE	MVNS ALL
	Type of VNS	NONE	Multi-step	NONE	Multi-step
	Search depth	NONE	4-5-6	NONE	4-5-6
	When	NONE	After 15-25-35	NONE	after 15-25-35
STOPPING CRITERION	***Type***	Dynamic	Dynamic	Dynamic	Dynamic
	When	after 50	after 50	after 50	after 50
AVERAGE PER CENT DEV (5 seed averaged)		26.81	3.10	72.23	11.54
AVERAGE PER CENT DEV (minimum)		24.33	3.01	60.85	11.08
AVERAGE CPU (Entire search-5 seed averaged)		110.58	286.35	64.24	101.96
AVERAGE CPU (Time elapsed up to best 5 seed averaged)		45.02	214.45	51.38	82.22

According to the results given in Table 5.29 and Table 5.30, intensification procedure again proves its necessity for the SS methodology. By applying a hybrid approach, the deviation value is decreased from 10.28 to 0.76 for 40-job 2-machine problem set, from 23.03 to 5.33 for 40-job 4-machine, from 26.81 to 3.10 for 60-job 2-machine problem set and finally the deviation value 72.23 reduced to 11.54 for the 60-job 4-machine problem. Similar to the HCSS used in SMTWT, the execution time of the

hybrid approach outperforms its candidate SS. Hence, the high computational times are still being observed. This is not a surprise for us, since the PMTT is more challenging problem than SMTWT. In PMTT, ready times, machine dependent process times and sequential setup times are all presented to reflect a real-life situation. These properties promote the solution space into a more complex environment which makes it harder to find an optimal solution in a shorter time.

Focusing on the performance of HCSS with respect to given problem sets, the reported results are quite satisfactory for 4-machine problems. The algorithm manages to solve optimally the 16.8 and 13.2 of the 20 instances for 40-job and 60-job problem sets respectively. However, the obtained average per cent deviation values appear higher when compared to 2-machine cases. This is due to the fact that the optimal total tardiness values corresponding to some instances of 4-machine problems are very low; thus small deviations between the found and optimal total tardiness values result in great average per cent deviations. (See Table F.8 and Table F.9). The more challenging problems are 2-machine problems especially the one containing 60-jobs. From the Table 5.29 and Table 5.30, the average per cent deviations for 40-job and 60-job problem sets are 0.76 and 3.10 respectively. If the best performing seed is considered, then these deviations become 0.43 for 40-jobs and 3.01 for 60-job problem set. These results seem satisfactory for our HCSS approach when compared to normal SS approach. As a remark, the HCSS algorithm outperforms some of the previous best known solutions found by Bilge et al. (2004) (See Appendix F.2). Therefore, our approach can be considered as a promising solution tool the tardiness related scheduling problems.

In the next section, the conclusive remarks are stated about the HCSS approach. The solution encoding, basic fundamentals, the driven ideas, advantages and disadvantages of HCSS algorithm are all discussed to provide a road book for future studies.

6. CONCLUSION

This study was designed to attack Single Machine Total Weighted Tardiness and Parallel Machine Total Tardiness problems by employing a hybrid meta-heuristic approach and to evaluate its performance. The Scatter Search and Variable Neighborhood Search approaches are studied and tailored to meet the theoretical requirements of the problems under study. Throughout the study, the solutions to the Single Machine Total Weighted Tardiness problem (SMTWT) and the Parallel Machine Total Tardiness problem (PMTT) are investigated. Performance criterion adopted for the SMTWT is to minimize the total weighted tardiness of all jobs and similarly for the PMTT, the aim is to diminish the total tardiness. The problem sets and corresponding best-known solutions employed in this investigation are adopted from the literature (Crauwels et al., 1998, Sivrikaya-Şerifoğlu, 1999).

While evolving our algorithm, numerical studies are conducted for each distinct solution strategy. Since the total number of combinatorial strategy approaches to be tested is very large, a sequential experimentation procedure is adopted. First, the HCSS approach is implemented in its most elementary form, which we call the basic algorithm. Then this basic algorithm is developed into its ultimate form by adapting strategies according to the experimentation results. At each experiment, we test one or more parameters/methods and select one (or sometimes more) level that perform at least as good as the others for each instance, and go on with this revised form of the algorithm.

For the SMTWT case, three types of problem sets are encountered, one with 40-jobs, the second with 50-jobs and the last with 100-jobs. Each of these sets involves 125 instances. Each instance is associated with a distinct process time, weight and due date. All these alternative problem sets are studied throughout this study and the results are obtained as: For the 40-job SMTWT problem, when the performance of five seeds is averaged, HCSS succeeds to have a 0.03 average per cent deviation from the best-known solution for each instance. If the best performing seed is taken into account, then this percentage decreases to 0.01 per cent. Similarly, for the 50-job problem HCSS obtains 0.11 average per cent deviation for 125 instances. If the best performing seed is considered, then this

percentage reduces down to 0.09 per cent. Finally, for the 100-job problem HCSS manages to attain average 0.50 per cent deviation from the best known solutions found in the literature. The best performing seed has a success 0.44 per cent.

For the PMTT case, the analyzed problem is formulated as follows. Four types of problems are encountered with two main job sets: one with 40-jobs with 2 or 4 machines in parallel, the second with 60-jobs with 2 or 4 machines in parallel. Each of these operations involves 20 instances. Each instance is associated with distinct process times, ready times, sequence dependent setup times and due dates. Attained results are given as: For the 40-job 2 machine case, HCSS achieves average deviation of 20 instances from the best known solution is at 0.76 per cent. If the best performing seed is taken into account, then this percentage goes down to 0.43 per cent. For the 40-job 4 machine case, average deviation of 20 instances from the best known solution is at 5.33 per cent. If the best performing seed is considered account, then this percentage is obtained as 4.73 per cent. For the 60-job 2 machine case, HCSS finds an average deviation of 20 instances from the best known solution at 3.10 per cent. If the best performing seed is taken into account, then this percentage reduces to 3.01 per cent. Finally the 60 job 4 machine case is studied. Our algorithm HCSS achieves an average deviation 11.54 per cent from the best known solution. If the best performing seed is encountered, the average per cent deviation is at 11.08 per cent.

Scatter Search is an emerging meta-heuristic approach in scheduling era. The developed methods are limited and there is not a huge source of unique procedures for tardiness related scheduling problems. Hence, it was not possible to use the available methods directly in all chosen problem domains. The strategies, definitions and methodologies explained in previous chapters are either amendments to existing methods or most of the time, original procedures developed to provide a different point of view for HCSS approach when it becomes incapable of effectively solving the problems at hand by using standard methods. The most noteworthy of these new strategies tested are:

- The use of a continuous solution encoding for a scheduling problem which is usually represented in discrete fashion.

- The use of an adaptive strategy in the solution combination by dynamically varying α value which determines the flexibility in creating an offspring.
- The distance measures proposed for measuring diversity.
- The idea of using multiple distance measures in selecting diverse solutions.

First of all, the solution space is considered as a continuous space and a real number encoded vector representation is introduced to map each and every solution hidden in the this space. Since Scatter Search uses weighted linear combinations of several solutions to produce new solutions; by employing linear combinations of start times, we express complex neighborhoods of a considered sequence in the search space. This also means that a given permutation can be reached from many other points in the continuous search space and allows an interesting flexibility property in terms of neighborhoods defined on the solution space. This new approach is supported with BLX-α operator which is one of the most effective combination methods developed for real-coded continuous GAs and with an alpha strategy which depends on dynamically changing α values. These methods favor the production of diversity in the population and prevent premature convergences. A similar representation was introduced by Bean (1994) where random numbers from zero to one are employed to encode a solution. These values are used as sort keys to determine a sequence of activities. However, in order to evaluate the performance of real number encoding, same study should be conducted with a proper discrete encoding where solutions are represented by permutation sequences and obtained results should be compared with each other.

As mentioned before, reference set plays the key role in SS method. All new solutions are originating from the combinations of elite and diverse solutions which are stored in this set. Therefore, the reference set should be constructed wisely to meet the demand for both high quality and dissimilar new solutions that are located at far regions of the solution space. In order to fulfill the mentioned requirement, an advanced reference set update method is developed in this study, which is feed from different sources. For updating best solutions, a reserve list is introduced as a long-term memory, which keeps the best found solutions during the entire search. Thus, the elite elements of the reference set are selected from both the reserve list and the generated new offspring. In addition to that, two different distance measurement methods, namely start and rank distance, are designed to select diverse solutions. These methods are new and have never been used

before as a diverse solution selection method in SS. When their performances are compared, the cooperation of both methods outperforms their individual performances, which is consistent with the idea behind constructing a set of dissimilar solutions. To sum up, it is proven that reference should be updated by using various solution selection procedures which leads to different sources of the search space.

Another conclusive remark should be stated for initial solution generation method. After implementing HCSS approach over different structured initial solution sets, it is clearly observed that seeded initial populations provide a better start point for the algorithm with respect to their randomly generated candidates. These seeded solutions are obtained by employing list heuristics as cited in the literature.

As a final remark, it is noteworthy that without an intensification strategy, the capabilities of Scatter Search are very limited. In its most basic form, SS converges to a poor local optimum or cycles around the same diverse solutions. To compensate this handicap, BLX-α method and dynamic alpha strategies are adapted to our HCSS approach. But, these methods alone are not sufficient to come up with superior optimum results. Therefore an intensification strategy, which implements a VNS to explore the neighborhoods of promising solutions, is integrated to our approach. The results with and without an intensification phase are so obvious that HCSS outperforms normal SS in every circumstances. One main disadvantageous aspect of the intensification strategy is the high execution time. This is mostly due to the fact that a high level programming tool, MATLAB, is used instead of a lower language and the code is not optimized in terms of time efficiency. The observed high computational times are also due to the intensive effort required for local improvement. This can be observed apparently in PMTT problems where we employ a three step VNS. For instance, when the optimal solution performance and the corresponding computation times of the HCSS method and related other algorithms are compared for the SMTWT problem, it is observed that the HCSS method tends to satisfactory results with reasonable execution time. When the performance of HCSS method is investigated for the PMTT problem, the computation time is relatively longer. The intensification strategy employed for the PMTT problem leads to repetitive VNS steps of different types. In addition to that algorithm is terminated after implementing all VNS

types while achieving no improvement during consecutive runs. Hence, the over-intensive effort results in computational overburdening and loss of time.

As a future study, a control mechanism to adaptively change the size of the reference set with respect to problem size can be devised. Because the HCSS approach with is not quite robust with respect to the size of the problem when attacked with fixed number of best and diverse solutions. The reference set used for small scale problems becomes in capable of covering huge solution space generated by a greater problem size. This situation is observed for SMTWT problem where the average per cent deviation value for 100-job problem set is higher when compared to 40-job and 50-job problem sets. Another modification can be investigated for the intensification strategy. Instead of exploring the whole neighborhood of a solution, a screening mechanism such as the candidate-list strategies employed by TS approach can be developed to decide promising neighborhoods. Also dynamically changing adaptive search depths can be utilized for VNS method.

To summarize, our main aim in this thesis has been to evaluate the performance of SS with respect to TS, GA and their hybrids which already produced highly satisfactory results in scheduling problems. To this end, based on our observations over two very hard tardiness based scheduling problems, it can be said that although SS is a promising strategy to elaborate, enchantments are required for it to cope-up with the aforementioned meta-heuristic approaches.

APPENDIX A: START DISTANCE MEASUREMENT METHOD

The start distance measurement method is demonstrated on an example where eight jobs are scheduled on a single machine. The reference set contains three best and three diverse solutions. The best solutions were already selected and the next step is to choose diverse solutions from a POOL including five different candidate solutions. Best solutions and candidate solutions are given in Table A.1.

Table A.1. Best solutions of reference set and candidate solutions in the pool

		START TIMES								FITNESS
BEST	1	1124	1492	1725	687	1530	1118	1106	993	***100***
	2	1135	1455	1664	632	1493	1005	993	1047	***137***
	3	1271	1457	1762	687	1567	1118	1106	993	***254***
CANDIDATE	1	1837	1385	1885	1274	1569	892	1530	898	***312***
	2	770	1898	1187	818	682	76	139	393	***568***
	3	175	1834	55	106	1138	866	139	1050	***714***
	4	1535	794	502	469	381	0	1431	1810	***403***
	5	483	761	599	199	799	1612	345	111	***689***

Take two solution, one from reference set and one from POOL .The start distance is calculated by using the equation;

$$\text{dist} = |(st_{job\ 1})_1 - (st_{job\ 1})_2| + \ldots + |(st_{job\ j})_1 - (st_{job\ j})_2| + \ldots + |(st_{job\ n})_1 - (st_{job\ n})_2|$$

Distance between best solution one & candidate solution one is calculated as follows;

$$\text{dist} = |1124-1837| + |1492-1385| + |1725-1885| + |687-1274| + |1530-1569| + |1118-892| + |1106-1530| + |993-898| = 2351$$

Using the same equation, the distances of other solution combinations are measured;

dist. (best 1 & candidate 2) = 4886

dist. (best 1 & candidate 4) = 5959

dist. (best 2 & candidate 1) = 2510

dist. (best 3 & candidate 5) = 6040

The calculated distance values and minimum of each row are stored in a distance matrix shown in Table A.2.

Table A.2. Measured distances between best & candidate solutions

		BEST			ROW MIN
		1	2	3	
CANDIDATE	1	2351	2510	2095	2095
	2	4886	4719	5142	4719
	3	5210	4825	5466	4825
	4	5959	5704	5851	5704
	5	5854	5729	6040	5729

Candidate solution # 5 corresponding to the maximum of the row minimums (5729) is promoted to the reference set as the diverse solution # 1 and deleted from the POOL. Since candidate solution # 5 is selected, the fifth row of the distance matrix is deleted and a column is inserted next to best solution # 3. Then, only the distances between diverse solution # 1 and remaining candidate solutions are measured and matrix is updated as shown in Table A.3.

Table A.3. Updated distance matrix

		BEST			DIV.	ROW MIN
		1	2	3	1	
CAND.	1	2351	2510	2095	7801	2095
	2	4886	4719	5142	4772	4719
	3	5210	4825	5466	4248	4248
	4	5959	5704	5851	6267	5704

According to maximum of the row minimums rule, candidate solution # 4 is selected as the diverse solution # 2. Fourth row is deleted and a new column is inserted. The updated distances are listed in Table A.4.

Table A.4. Updated distance matrix

		BEST			DIV.		ROW MIN
		1	2	3	1	2	
CAND.	1	2351	2510	2095	7801	6172	2095
	2	4886	4719	5142	4772	5989	4719
	3	5210	4825	5466	4248	6865	4248

Candidate solution # 2 is decided as the last diverse solution for the reference set. The reference set is updated with new diverse solutions and take the final structure shown in Table A.5.

Table A.5. Final reference set

		START TIMES								FITNESS
BEST	1	1124	1492	1725	687	1530	1118	1106	993	***100***
	2	1135	1455	1664	632	1493	1005	993	1047	***137***
	3	1271	1457	1762	687	1567	1118	1106	993	***254***
DIVERSE	1	770	1898	1187	818	682	76	139	393	***568***
	2	1535	794	502	469	381	0	1431	1810	***403***
	3	483	761	599	199	799	1612	345	111	***689***

APPENDIX B: RANK DISTANCE MEASUREMENT METHOD

The rank distance measurement method is demonstrated on an example problem aiming to schedule eight jobs on a single machine. The reference set contains three best and three diverse solutions. The best solutions were already selected and the next step is to choose diverse solutions from a POOL including five different candidate solutions. Best solutions and candidate solutions are given in Table B.1.

Table B.1. Best solutions of reference set and candidate solutions in the pool

		START TIMES								FITNESS
BEST	1	1124	1492	1725	687	1530	1118	1106	993	***100***
	2	1135	1455	1664	632	1493	1005	993	1047	***137***
	3	1271	1457	1567	687	1762	1118	993	1106	***254***
CANDIDATE	1	1837	1385	1885	1274	1569	892	1530	898	***312***
	2	770	1898	1187	818	682	76	139	393	***568***
	3	175	1834	55	106	1138	866	139	1050	***714***
	4	1535	794	502	469	381	0	1431	1810	***403***
	5	483	761	599	199	799	1612	345	111	***689***

Rank distance measurement method is constructed on a simple rearrangement procedure where places of numbers are changed with restricted insertion moves aiming to catch a target sequence of these numbers. Our solutions contain start time and fitness values which should be revised to obtain meaningful number sequences. Therefore, start times are sorted in non-decreasing order and corresponding job numbers are noted in same order. The sorted jobs numbers for each job is given in Table B.2.

Table B.2. Rank matrix

		SORTED JOB NUMBERS								FITNESS
BEST	1	4	8	7	6	1	2	5	3	***100***
	2	4	7	6	8	1	2	5	3	***137***
	3	4	7	8	6	1	2	3	5	***254***
CANDIDATE	1	6	8	4	2	7	5	1	3	***312***
	2	6	7	8	5	1	4	3	2	***568***
	3	3	4	7	1	6	8	5	2	***714***
	4	6	5	4	3	2	7	1	8	***403***
	5	8	4	7	1	3	2	5	6	***689***

In order to measure rank distance between two solutions, one solution is taken from POOL and one from the reference set which is denoted as target sequence. The total number of restricted insertion moves utilized to convert candidate sequence into the target sequence is the rank distance between these two. A restricted insertion move allows only insertion of a job between its two preceding jobs. For example, the rank distance between best solution # 1 & candidate solution # 1 is evaluated as follows;

The target sequence = *4* *8* *7* *6* *1* *2* *5* *3*

The candidate sequence = *6* *8* *4* *2* *7* *5* *1* *3*

Start with the job # 4, two insertion moves are needed to place this job into the first place as shown below.

The candidate sequence = *4* *6* *8* *2* *7* *5* *1* *3*

Next, job # 8 is set to target position with one insertion move.

The candidate sequence = *4* *8* *6* *2* *7* *5* *1* *3*

For job # 7, two insertion moves is needed.

The candidate sequence = *4* *8* *7* *6* *2* *5* *1* *3*

And after placing job # 1 into its right place by performing two moves, the target sequence is obtained.

The candidate sequence = *4* *8* *7* *6* *1* *2* *5* *3*

The total seven insertion moves is needed to convert candidate into target sequence. This rank distance is then stored in a distance matrix. The calculated distance values and minimum of each row are stored in a distance matrix shown in Table B.3.

Table B.3. Distance matrix

		BEST			ROW MIN
		1	2	3	
CANDIDATE	1	7	7	9	7
	2	11	9	11	9
	3	12	10	10	10
	4	17	15	17	15
	5	7	9	7	7

Candidate solution # 4 corresponding to the maximum of the row minimums (15) is promoted to the reference set as the diverse solution # 1 and deleted from the POOL. Since candidate solution # 4 is selected, the fourth row of the distance matrix is deleted and a column is inserted next to best solution # 3. Then, only the distances between diverse solution # 1 and remaining candidate solutions are measured and matrix is updated as shown in Table B.4.

Table B.4. Updated distance matrix

		BEST			DIV.	ROW MIN
		1	2	3	1	
CAND.	1	7	7	9	12	7
	2	11	9	11	12	9
	3	12	10	10	13	10
	5	7	9	7	22	7

According to maximum of the row minimums rule, candidate solution # 3 is selected as the diverse solution # 2. Third row is deleted and a new column is inserted. The updated distances are listed in Table B.5.

Table B.5. Updated distance matrix

		BEST			DIV.		ROW MIN
		1	2	3	1	2	
CAND.	1	7	7	9	12	17	7
	2	11	9	11	12	15	9
	5	7	9	7	22	12	7

Candidate solution # 2 is decided as the last diverse solution for the reference set. The reference set is updated with new diverse solutions and take the final structure shown in Table B.6.

Table B.6. Final reference set

		START TIMES								FITNESS
BEST	1	1124	1492	1725	687	1530	1118	1106	993	***100***
	2	1135	1455	1664	632	1493	1005	993	1047	***137***
	3	1271	1457	1567	687	1762	1118	993	1106	***254***
DIVERSE	1	1535	794	502	469	381	0	1431	1810	***403***
	2	175	1834	55	106	1138	866	139	1050	***714***
	3	770	1898	1187	818	682	76	139	393	***568***

APPENDIX C: SOLUTION COMBINATION METHOD

Let us assume that, the subset generation method produces a subset of two solutions *X* and *Y* as a basis for creating combined solution. In order to illustrate the methodology, we provide hypothetical process times, due dates, weights for each job and start times and total tardy values for *X* and *Y* as shown below;

Table C.1. Process time, weight and due date of jobs #1,….,8

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***
pt	48	38	51	33	88	6	12	88
wt	4	3	5	8	2	3	8	3
dt	168	315	220	248	151	10	40	33

Table C.2. Two parent solutions

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***	***FIT***
X	139	187	225	0	276	133	33	45	**680**
Y	191	323	272	239	103	0	3	15	**376**

The start time of each job for the new solution will be a randomly selected number from an interval determined by the combination of parent solutions. BLX-α combination method generates an offspring $Z=(Z_{job\#1},\ldots, Z_{jobni})$ where $Z_{job\#i}$ is a randomly chosen number of the interval $[c_{min} - I.\alpha,\ c_{max} + I.\ \alpha]$ where $c_{max}=max(X_{job\#i},\ Y_{job\#i})$, $c_{min}=min(X_{job\#i},\ Y_{job\#i})$ and $I= c_{max} - c_{min}$ and α is a constant value.

Starting with the job#1;

$X_{job\#1} = 139,\ Y_{job\#1} = 191,$

$c_{max} = 191,\ c_{min}= 139$ *and* $I =(191 – 139) = 52$

interval $= [139 – (52x0.5),\ 191+(52x0.5)] = [113, 217]$ // $\alpha = 0.5$

$Z_{job\#1}$ is a randomly selected real number from the interval [113, 217]. Suppose that imaginary start time of $Z_{job\#1}$ is chosen as 149.8.

Continuing with job#2;

$X_{job\#1} = 187,\ Y_{job\#1} = 323,$

$c_{max} = 323,\ c_{min} = 187$ *and* $I = (323 - 187) = 136$

interval $= [187 - (136x0.5),\ 323+(136x0.5)] = [119, 391]$

$Z_{job\#2}$ is randomly selected as 380.3. The methodology repeats itself until all the start time values are determined. The offspring with temporary start time values is a trial solution. An improvement method is applied to correct start time values and obtain the total tardiness value of the improved solution.

Table C.3. Offspring solution before improvement method

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***
Z	149.8	380.3	233.7	168.4	301.1	10	15.9	33

Improvement method is a simple algorithm that sorts jobs according to their increasing start times and then schedule them on a single machine while computing their real start times and tardy values. Finally, existing values of the pre improved solution are replaced with real values and fitness is inserted into the offspring.

Improvement method

sort Z in order of non-decreasing start times

sorted Z = 10 15.9 33 149.8 168.4 233.7 301.1 380.3

sorted jobs = 6 7 8 1 4 3 5 2

calculate start time and tardy for each job j, // j=1,…., 8

job #6 : start time = 0, completion time = 6 , tardy = 0

job #7 : start time = 6, completion time = 18 , tardy = 0

job #8 : start time = 18, completion time = 106 , tardy = 219

job #1 : start time = 106, completion time = 154 , tardy = 0

job #4 : start time = 154, completion time = 187 , tardy = 0

job #3 : start time = 187, completion time = 238 , tardy = 90

job #5 : start time = 238, completion time = 326 , tardy = 350

job #2 : start time = 326, completion time = 364 , tardy = 147

rewrite Z with updated start times and total tardy value

updated Z = 106 326 187 154 238 0 6 18 806

Figure C.4. Illustration of improvement method

APPENDIX D: VARIABLE NEIGHBORHOOD SEARCH - SMTWT

The VNS methodology employed in our HCSS approach will be discussed in the following section. Initially, VNS used in SMTWT problem is investigated and neighborhood generation, neighborhood search and evaluation step are all demonstrated by using a sample problem. Our sample problem consists of eight jobs with distinct process times, weights and due dates and ready at time zero to be scheduled on a single machine. The aim here is to minimize total weighted tardiness value. The process times, weights and due dates are given in Table D.1. Before applying VNS, HCSS comes up with an elite solution *Z* such that *Z* = [106 326 187 154 238 0 6 18 806]. The first eight entities are start times of job *j (j = 1,…, 8)* and final one is the *twt* value.

Table D.1. Process time, weight and due date of jobs

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***
pt	48	38	51	33	88	6	12	88
wt	4	3	5	8	2	3	8	3
dt	168	315	220	248	151	10	40	33

VNS 2-3-4 is applied to generate neighborhood of current solution *Z*. The sorted *Z* sequence is divided into groups, each having number of jobs determined by search depth. At first, our search depth is equal to two. Hence, four groups with two jobs are rearranged to create new neighborhoods.

sorted Z = 6 7 8 1 4 3 5 2
group #1 = (6 7), group #2 = (8 1), group #3 = (4 3), group #4 = (5 2).

Starting with the first group, all alternative permutations are written

alternative#1 = (6 7), alternative#2 = (7 6)

Then, first group jobs in the sorted *Z* are replaced with one of the alternative permutations such that

neighbor # 0 = 6 7 8 1 4 3 5 2
neighbor # 1 = 7 6 8 1 4 3 5 2

The new neighbors are evaluated by scheduling jobs using their processing times, weights and due dates and the final fitness values are computed by summing all non-zero tardiness values associated to considered sequence.

fitness of neighbor # 0 = 806
fitness of neighbor # 1 = 830

These fitness values are then compared with current solution total weighted tardiness value. If one of neighbor is superior to the current one, it becomes the new current solution and sorted *Z* is updated accordingly. In our case, none of the neighbors is better than the present schedule. Therefore, the sorted *Z* is kept without altering. Moving to next group;

alternative#1 = (8 1), alternative#2 = (1 8)
neighbor # 0 = 6 7 8 1 4 3 5 2
neighbor # 1 = 6 7 1 8 4 3 5 2

and evaluating their fitness;

fitness of neighbor # 0 = 806
fitness of neighbor # 1 = 950

the current solution is conserved and new neighbors are generated by using group three

neighbor # 0 = 6 7 8 1 4 3 5 2
neighbor # 1 = 6 7 8 1 3 4 5 2

and their fitness values;

fitness of neighbor # 0 = 806

fitness of neighbor # 1 = 716

Therefore, neighbor #1 becomes new current solution. Instead of going back and dividing the new sequence into new groups, we continue with the last group;

neighbor # 0 = 6 7 8 1 3 4 5 2
neighbor # 1 = 6 7 8 1 3 4 2 5

corresponding fitness values;

fitness of neighbor # 0 = 716
fitness of neighbor # 1 = 645

According to this result, our new best order is [6 7 8 1 3 4 2 5]. After exploring all neighbors, we take the search more further distances. The search depth is set to three, and the new groups are formed starting from the very end of the sequence.

sorted Z = 6 7 8 1 3 4 2 5
group #1 = (8 1 3), group #2 = (4 2 5)

Starting with the first group, all alternative permutations are written

(8 1 3), (8 3 1), (1 8 3), (1 3 8), (3 1 8), (3 8 1)

and neighbors of the current solution is generated as;

neighbor # 0 = 6 7 8 1 3 4 2 5
neighbor # 1 = 6 7 8 3 1 4 2 5
neighbor # 2 = 6 7 1 8 3 4 2 5
neighbor # 3 = 6 7 1 3 8 4 2 5
neighbor # 4 = 6 7 3 1 8 4 2 5
neighbor # 5 = 6 7 3 8 1 4 2 5

total weighted tardiness values are computed for each permutation of group #1

fitness of neighbor # 0 = 645
fitness of neighbor # 1 = 793
fitness of neighbor # 2 = 789
fitness of neighbor # 3 = 942
fitness of neighbor # 4 = 942
fitness of neighbor # 5 = 798

considering the next group;

(4 2 5), (4 5 2), (2 4 5), (2 5 4), (5 2 4), (5 4 2)
neighbor # 0 = 6 7 8 1 3 4 2 5
neighbor # 1 = 6 7 8 1 3 4 5 2
neighbor # 2 = 6 7 8 1 3 2 4 5
neighbor # 3 = 6 7 8 1 3 2 5 4
neighbor # 4 = 6 7 8 1 3 5 2 4
neighbor # 5 = 6 7 8 1 3 5 4 2

The corresponding fitness values are

fitness of neighbor # 0 = 645
fitness of neighbor # 1 = 716
fitness of neighbor # 2 = 869
fitness of neighbor # 3 = 862
fitness of neighbor # 4 = 834
fitness of neighbor # 5 = 629

Neighbor # 5 is superior to both the current solution and the other neighbors. Hence, the best sequence becomes [6 7 8 1 3 5 4 2] and the search depth is decreased to two. The search procedure continues as stated above until the termination criterion is met.

APPENDIX E: VARIABLE NEIGHBORHOOD SEARCH – PMTT

The three different type VNS techniques employed in our HCSS approach will be discussed in the following section. Initially, type #1 VNS technique used in PMTT problem is investigated and neighborhood generation, neighborhood search and evaluation step are all demonstrated by using a sample problem. Our sample problem consists of eight jobs with distinct process times, ready times, sequence dependent setup times and due dates. The two machines utilized in the problem are not identical, thus the process times and setup times related to same jobs are varying depending on the machine. The aim here is to minimize total tardiness value. The ready times and due dates are given in Table E.1 and process times and sequence dependent setup times are presented in Table E.2 and Table E.3. Before applying type #1 VNS, HCSS comes up with an elite solution *Z* such that;

$$Z = [32\ \ 43\ \ 31\ \ 24\ \ 37\ \ 11\ \ 18\ \ 50\ 1\ \ 2\ \ 2\ \ 1\ \ 1\ \ 1\ \ 2\ \ 2\ \ 26]$$

where the first eight entities are start times of job *j (j = 1,…, 8)*, the second eight entities are the machine indices and final one is the total tardiness value.

Table E.1. Ready time and due date of jobs

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***
rt	27	43	31	24	30	11	18	35
dt	32	51	38	34	36	20	31	46

Table E.2. Process times and setup times for machine #1

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***
Job#1	0	1	3	2	3	1	1	2
Job#2	2	0	2	3	1	2	3	1
Job#3	1	1	0	1	2	3	1	1
Job#4	3	2	3	0	2	1	2	2
Job#5	2	2	1	3	0	1	3	1
Job#6	2	1	1	1	2	0	3	3
Job#7	3	1	2	1	3	1	0	3
Job#8	1	2	1	3	2	1	1	0
pt	*2*	*5*	*4*	*7*	*3*	*6*	*10*	*8*

Table E.3. Process times and setup times for machine #2

	Job#1	***Job#2***	***Job#3***	***Job#4***	***Job#5***	***Job#6***	***Job#7***	***Job#8***
Job#1	0	3	1	2	1	3	1	1
Job#2	2	0	1	3	2	1	3	1
Job#3	1	1	0	1	2	3	1	1
Job#4	1	2	1	0	2	1	1	3
Job#5	1	3	2	3	0	1	1	2
Job#6	2	2	1	3	1	0	3	1
Job#7	2	3	2	3	1	2	0	1
Job#8	2	1	1	1	2	3	3	0
pt	*3*	*6*	*5*	*8*	*4*	*7*	*11*	*9*

In the type #1 VNS technique, the sorted job sequence of *Z* is recorded and kept fixed during the entire neighborhood search. Then this sequence is divided into groups containing same number of jobs defined by VNS depth and their corresponding machine assignments. Starting with the first group, all possible machine assignments are listed for this group's jobs without altering the job sequence and remaining machine indices. VNS 2-3-4 is applied to generate neighbors of current solution *Z*. The sorted *Z* is as follows;

sorted Z =[6 7 4 3 1 5 2 8 1 2 1 2 1 1 2 2]
group #1=(6 7 1 2), group #2=(4 3 1 2), group #3=(1 5 1 1), group #4=(2 8 2 2)

Starting with the first group, all alternative machine assignments are written as

alt. #1 = (6 7 1 1), alt. #2 = (6 7 2 1), alt. #3 = (6 7 2 2)

Observing the alternative #1, the processing order of job #6 and job #7 does not change. The only difference is that the both jobs will be process on machine #1 whereas they are initially processed by machine #1 and machine #2 respectively. Inserting alternatives into the initial sorted sequence where all are representing a different neighborhood of the current solution;

neighbor # 1 = [6 7 4 3 1 5 2 8 1 1 1 2 1 1 2 2]
neighbor # 2 = [6 7 4 3 1 5 2 8 2 1 1 2 1 1 2 2]
neighbor # 3 = [6 7 4 3 1 5 2 8 2 2 1 2 1 1 2 2]

The new neighbors are evaluated by scheduling jobs using their machine dependent process times, setup times, ready times and due dates. Then the final fitness values are computed by summing all non-zero tardiness values associated to considered sequence.

fitness of neighbor # 1 = 45
fitness of neighbor # 2 = 36
fitness of neighbor # 3 = 26

These fitness values are then compared with current solution total tardiness value. If one of neighbor solution is superior to the current one, it becomes the new current solution and sorted *Z* is updated accordingly. In our case, the neighbor #3 has the value 26 which is not better than the present best solution. Therefore, the sorted *Z* is kept without altering and VNS moves to next group. The next group and corresponding neighbors are

group #2 = (4 3 1 2) alt. #1 = (4 3 1 1), alt. #2 = (4 3 2 1), alt. #3 = (4 3 2 2)
neighbor # 1 = [6 7 4 3 1 5 2 8 1 2 1 1 1 1 2 2]
neighbor # 2 = [6 7 4 3 1 5 2 8 1 2 2 1 1 1 2 2]
neighbor # 3 = [6 7 4 3 1 5 2 8 1 2 2 2 1 1 2 2]

and evaluating their fitness;

fitness of neighbor # 1 = 40
fitness of neighbor # 2 = 37
fitness of neighbor # 3 = 34

According to the results, the current solution is conserved. Considering the next group;

group #3 = (1 5 1 1) alt. #1 = (1 5 1 2), alt. #2 = (1 5 2 1), alt. #3 = (1 5 2 2)
neighbor # 1 = [6 7 4 3 1 5 2 8 1 2 1 2 1 2 2 2]
neighbor # 2 = [6 7 4 3 1 5 2 8 1 2 1 2 2 1 2 2]
neighbor # 3 = [6 7 4 3 1 5 2 8 1 2 1 2 2 2 2 2]

and their fitness values are

fitness of neighbor # 1 = 32
fitness of neighbor # 2 = 28
fitness of neighbor # 3 = 46

No improvement can be achieved, thus VNS explores the neighbors of the last group;

group #3 = (2 8 2 2) alt. #1 = (2 8 1 1), alt. #2 = (2 8 1 2), alt. #3 = (2 8 2 1)
neighbor # 1 = [6 7 4 3 1 5 2 8 1 2 1 2 1 1 1 1]
neighbor # 2 = [6 7 4 3 1 5 2 8 1 2 1 2 1 1 1 2]
neighbor # 3 = [6 7 4 3 1 5 2 8 1 2 1 2 1 1 2 1]

The corresponding fitness values are

fitness of neighbor # 1 = 25
fitness of neighbor # 2 = 14
fitness of neighbor # 3 = 18

Neighbor #2 results with a fitness value 14 which is superior to best current solution. Hence, the new sorted *Z* is updated as follows

sorted Z = [6 7 4 3 1 5 2 8 1 2 1 2 1 1 1 2]

After exploring all neighbors, we take the search a one step further distance. The search depth is set to three, and the new groups are formed starting from the very end of the sequence. Here, the type #1 VNS technique will not be demonstrated with its all steps and not be concluded. The investigating first group alternatives for search depth three gives sufficient information to understand the further execution of basic the methodology behind our type #1 VNS technique. The groups and corresponding neighbors for the first one are given below;

group #1 = (4 3 1 1 2 1) group #2 = (5 2 8 1 1 2)
alt. #1 =(4 3 1 1 1 1) alt. #2 =(4 3 1 1 1 2) alt. #3 =(4 3 1 1 2 2)

alt. #4 =(4 3 1 2 1 1) alt. #5 =(4 3 1 2 2 1) alt. #6 =(4 3 1 2 1 2)
alt. #7 =(4 3 1 2 2 2)

neighbor # 1 = [6 7 4 3 1 5 2 8 1 2 1 1 1 1 1 2]
neighbor # 2 = [6 7 4 3 1 5 2 8 1 2 1 1 2 1 1 2]
neighbor # 3 = [6 7 4 3 1 5 2 8 1 2 1 2 2 1 1 2]
neighbor # 4 = [6 7 4 3 1 5 2 8 1 2 2 1 1 1 1 2]
neighbor # 5 = [6 7 4 3 1 5 2 8 1 2 2 2 1 1 1 2]
neighbor # 6 = [6 7 4 3 1 5 2 8 1 2 2 1 2 1 1 2]
neighbor # 7 = [6 7 4 3 1 5 2 8 1 2 2 2 2 1 1 2]

and their fitness values are

fitness of neighbor # 1 = 27
fitness of neighbor # 2 = 11
fitness of neighbor # 3 = 17
fitness of neighbor # 4 = 29
fitness of neighbor # 5 = 25
fitness of neighbor # 6 = 31
fitness of neighbor # 7 = 46

According to the fitness values of the neighbor, machine assignment alternative #2 decreases the total tardiness value from 14 to 11. Therefore, sorted *Z* is updated and the new current solution becomes;

sorted Z = [6 7 4 3 1 5 2 8 1 2 1 1 2 1 1 2]

Our demonstration for the type #1 VNS ends up at this point, the remaining neighborhood generations and evaluations can be performed according to methodology described so far.

To illustrate type #2 VNS technique, the same problem set and the same VNS depth is utilized. Our elite solution Z is considered such that

Z = [29 44 32 24 39 11 18 35 2 1 1 1 1 1 2 2 11]

where the first eight entities are start times of job *j (j = 1,…, 8)*, the second eight entities are the machine indices and final one is the total tardiness value. Initially, the start times are ordered with respect non-decreasing values. Then without considering the machine indices our sorted *Z* vector is constructed. The methodology employed to construct neighborhood of a current solution is exactly the same that is used in SMTWT. The only difference arises at the evaluation step of the fitness value of the generated neighbors. Type #2 VNS technique assigns each job to a machine by using the EFT rule and then calculates its corresponding total tardiness value. This evaluation step is briefly described in the following example.

Suppose that we implement type #2 VNS with search depth two, but do not manage to find a better solution. Therefore, the size of the search depth is increased to three and the sequence is grouped as follows;

sorted Z = [6 7 4 3 1 5 2 8]
group #1 = (4 3 1) group #2 = (5 2 8)

The alternative permutation of group #1 jobs and corresponding neighbors are

(4 1 3), (1 3 4), (1 4 3), (3 1 4), (3 4 1)
neighbor # 1 = 6 7 4 1 3 5 2 8
neighbor # 2 = 6 7 1 3 4 5 2 8
neighbor # 3 = 6 7 1 4 3 5 2 8
neighbor # 4 = 6 7 3 1 4 5 2 8
neighbor # 5 = 6 7 3 4 1 5 2 8

Evaluating the fitness value of neighbor #1, the completion time of job #6 is calculated on both machines. And the job is assigned to the machine with the earlier completion time. The tardiness value of the current job is computed accordingly.

Completion time of job #6 on machine #1 = $\max(r_6^I, f^I) + p_6^I = 11+6 = 17$

Completion time of job #6 on machine #2 = $\max(r_6^{II}, f^{II}) + p_6^{II} = 11+7 = 18$

Then job #6 is dispatch to machine #1 and the finish time (f^I) for machine #1 is set to 17 and (f^{II}) for machine #2 to 0. The tardiness value is calculated using the formula $T_j = \max(0, C_j - d_j)$, i.e. max(0, 17-20) = 0. Continuing with job #7;

Completion time of job #7 on machine #1 = $\max(r_7^I, f^I) + p_7^I + s_{67}^I = 31$

Completion time of job #7 on machine #2 = $\max(r_7^{II}, f^{II}) + p_7^{II} = 29$

Job # 7 is then assign to the machine #2 and the tardiness value is $T_7 = \max(0, 29-31) = 0$. The finish time of machine #2 becomes 29. The remaining calculations are performed in same manner and not shown here. The final start times, machine indices and total tardiness value related to neighbor #1 is computed as

neighbor #1 = [29 43 32 24 34 11 18 39 2 1 1 1 2 1 2 2 10]

where the first eight entities are start times of job *j (j = 1,…, 8)*, the second eight entities are the machine indices and final one is the total tardiness value. The fitness values of the other neighbors are;

fitness of neighbor # 2 = 19
fitness of neighbor # 3 = 23
fitness of neighbor # 4 = 19
fitness of neighbor # 5 = 30

From the obtained fitness values, it is obvious that neighbor #1 outperforms the other candidates. Therefore, our sorted Z value becomes [6 7 4 1 3 5 2 8] and the current solution is updated as;

Z = [29 43 32 24 34 11 18 39 2 1 1 1 2 1 2 2 10]

For the *group #2 = (5 2 8)*, the permutation alternatives and generated neighbors are as follows;

(5 8 2), (8 2 5), (8 5 2), (2 5 8), (2 8 5)
neighbor # 1 = 6 7 4 1 3 5 8 2
neighbor # 2 = 6 7 4 1 3 8 2 5
neighbor # 3 = 6 7 4 1 3 8 5 2
neighbor # 4 = 6 7 4 1 3 2 5 8
neighbor # 5 = 6 7 4 1 3 2 8 5

and the corresponding fitness values are;

fitness of neighbor # 1 = 9
fitness of neighbor # 2 = 18
fitness of neighbor # 3 = 11
fitness of neighbor # 4 = 10
fitness of neighbor # 5 = 18

Again, neighbor #1 comes up with a better total tardiness value. Hence our sorted Z and new current solution vector is updated as follows;

sorted Z = [6 7 4 1 3 5 8 2]
Z = [29 43 32 24 34 11 18 39 2 2 1 1 2 1 2 1 9]

Since, the VNS improves the current solution, the search depth decreases the value two and the variable neighborhood search is continued by repeating the same methodology until the termination criterion is met i.e. the current solution can not be improved with maximum VNS depth. Finally, the type #3 VNS technique is described by using same problem set and VNS 2-3-4. The alternative job permutations and neighborhood generation method are exactly the same as the ones used in type #2 VNS. The only difference as we indicate before; is the evaluation procedure of a neighborhood solution. Previously, EFT rule is utilized to perform both machine assignment and total tardiness value computation. This time, a recursive rule which is explained in Section 4.2.4, is employed as a substitute for EFT rule. Therefore, instead of showing all the steps needed to generate a

neighborhood, we only illustrate the recursive rule step by step while evaluating a selected neighborhood.

Assume that, VNS comes up with a neighbor such that [6 7 4 1 3 5 8 2] and the corresponding sorted start times of jobs are [11 16 24 29 32 40 42 43]. At the beginning both machines' finish times (f^I and f^{II}) are zero and the process times and setup times are given in Table E.2 and Table E.3. Comparing job #6's start time with both of the finish times, it is seen that the job can be assign any of the machines. Thus, method constructs a branch indicating both of the alternatives as shown in Figure E.1. In figure, the expression *(X,Y)* located on the nodes indicates that job *X* is processed on machine *Y* and the value below the nodes is the total tardiness value of assignments performed so far

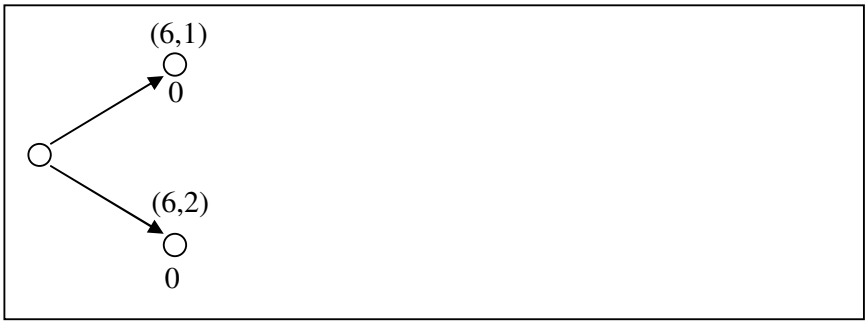

Figure E.1. Flow chart for recursive algorithm #1

Continuing with the upper branch, the first job is assigned to machine #1 and the f^I is found as 17 and f^{II} equals to zero. The start time of job #7 is 16 which is earlier than f^I but tardier than f^{II}. Hence, it is allocated to machine #2. The corresponding tardiness value of job #7 is 0 and the f^{II} becomes 29. The flow chart is given in Figure E.2.

The next one is job #4 with start time 24 which is earlier than both machines' finish times. So, it is assigned to the machine #1 which has the smallest finish time value. After performing required calculations, the tardiness of job #4 is found as 0 and the f^I increases to 32. Comparing f^I and f^{II} with job #1's start time (i.e. 29), the machine #2 is selected with the smaller finish time value. Then f^{II} becomes 34 and the tardiness value $T_1 = \max(0, C_1 - d_1) = 2$. Job #3 with start time 32 is allocated to machine #1 where start time of job #3 equals to f^I but smaller than f^{II} value. Calculating the tardiness value, it is observed that completion time of job #3 exceeds its due date by one unit. The next start time of job #5 is

40 where it is larger than both f^I = 39 and f^{II} = 34. Thus, it can be dispatched any of the machines which is shown by two branches in Figure E.3.

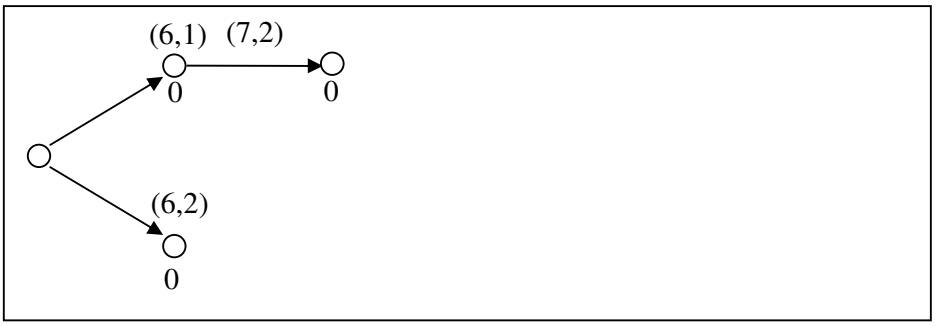

Figure E.2. Flow chart for recursive algorithm #2

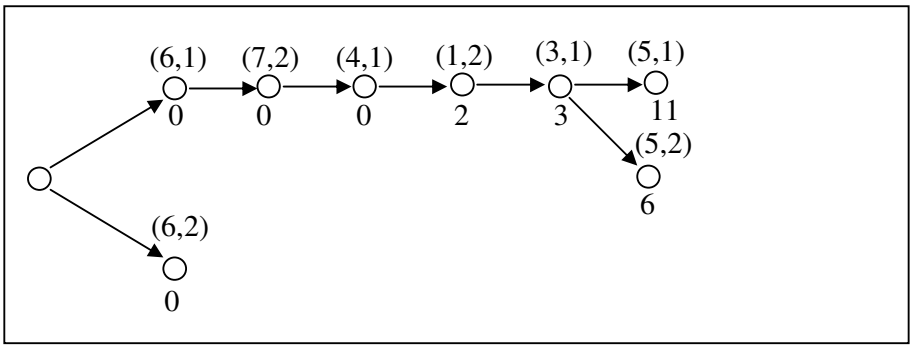

Figure E.3. Flow chart for recursive algorithm-step #3

For the upper branch where the job #5 is processed on machine #1 and the corresponding tardiness value is 8, f^I = 44 and f^{II} = 34. On the other hand, lower branch comes up with a tardiness value equals to 3, f^I = 39 and f^{II} = 39. Moving to the next step over the upper branch, the start time job #8 is compared with the finish times. Since start time 42 is less than f^I and more than f^{II}, machine #2 is employed for job #8. Then the results are; $T_8 = 0$, $f^I = 44$ and $f^{II} = 45$. The final one to be scheduled is the job #2. The start time 43 is earlier than both of the finish times. Therefore, machine #1 with the earlier finish time is assigned for the last job. The tardiness value related to this assignment is zero, so the total tardiness value becomes 11 for the entire sequence. This value is memorized as the best fitness value found so far and represents an upper bound for the total tardiness. When we reach to the end of lane, i.e. the allocation of all jobs is completed; we turn back to the previous node where the lane is divided into two branches. At this point, the unvisited branch is selected and the remaining assignments are performed accordingly. Any time, the

total tardiness value which is computed after dispatching a job, is greater than the upper bound, the further assignment related to this lane is not executed and the algorithm turns back to a previous node that is branched out.

Consider our example, after finishing the allocation, we move back to node (3,1) and select the lower branch and jumps to the node (5,2). Since the total tardiness value corresponding to this node is smaller than the upper bound, we continue with the next job, i.e. job #8. The start time of the job is 42, which is greater than both f^{I} and f^{II}. Thus, the lane is divided into two branches and two new nodes are placed at the end of each branch, which are indicating machine assignments and related fitness values.

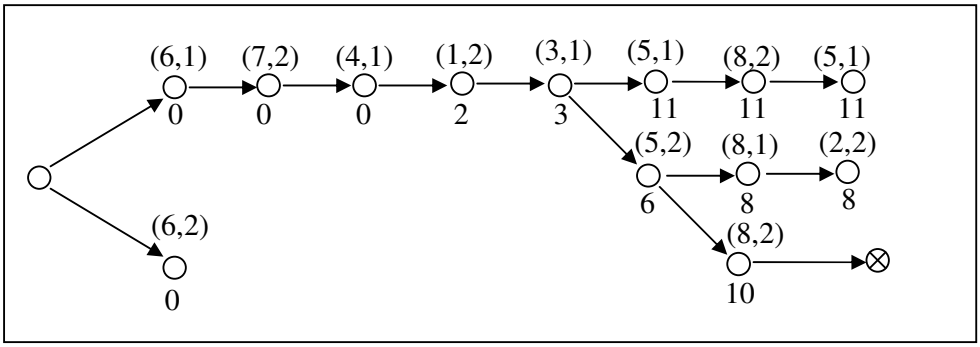

Figure E.4. Flow chart for recursive algorithm #4

The finish times and total tardiness for each node is calculated as; f^{I} = 48, f^{II} = 39 and T_8 = 2 for upper branch and f^{I} = 39, f^{II} = 50 and T_8 = 4 for the lower branch. Moving along the upper branch, the job #2 is assigned to the machine #2 where the start time 43 is greater than f^{II}. Then the total tardiness value for the entire sequence computed as eight which becomes the new upper bound. Turning back to the branched node and jumping to the point (8,2), we realize that the total tardiness value is greater than the upper bound. Therefore, there is no need to move further and the current lane is blocked as shown in Figure E.4. The remaining calculations are performed in the same way as described. The final upper bound is considered as the fitness value of the selected neighborhood.

When dealing with 4-machine PMTT problems, instead of constructing four branches that define each machine assignment separately, we only form two of them that correspond to the machines with smallest finish times. In other words, before allocating a

job, the four finish times are sorted and two of them are selected according to non-decreasing values. Then the start time of the job is compared with chosen f^m's where f^m is the finish time of machine m and the job is assigned to a machine depending on the selection procedure. The machine selection procedures of 2-machine problem can be applied for 4-machine case. Each time an assignment is performed, the finish times are updated and two smallest f^m's are selected accordingly. The remaining methodology is same as stated above.

APPENDIX F: BEST KNOWN SOLUTION TO THE PROBLEM SETS

This appendix provides the optimal/best-known solutions to the problem sets of tardiness related scheduling problems used for numerical experimentations throughout this thesis. These optimal/best-known solutions are obtained from the literature. In the first section, reported optimal/best-known solutions (Crauwels et al., 1998, Congram et al., 2002) for the SMTWT problem are given and followed by our final results for the five seed run. In the next section, best known solutions to PMTT problem sets are stated. Two set of best-known solutions are given in Appendix F.2, the first set contains the solutions obtained by Bilge et al. (2004) and the second set includes the updated best-known solutions reported by Anghinolfi and Paolucci (2006). This section also shows the best-found solutions to PMTT during the numerical studies conducted with HCSS approach and the min-max total tardiness values obtained by five seed final run for the each instance of the problem sets.

Appendix F.1. Best-Known Solutions to SMTWT

For the 40-job and 50-job problem sets, the optimal solutions are known for most of the instances. The exceptions are indicated. As for the 100-job problem set, due to the problem size limitations, only best-known solutions are reported. However, as these solutions have not been improved ever since they have been published, there is a great evidence that they are actually optimal.

Table F.1. Optimal/best-known solutions to the 40-job problem set – SMTWT

40-JOB PROBLEM SET									
Instance	***Optimal***	***Instance***	***Optimal***	***Instance***	***Optimal***	***Instance***	***Optimal***	***Instance***	***Optimal***
001	913	***026***	108	***051***	0	***076***	0	***101***	0
002	1225	***027***	64	***052***	0	***077***	0	***102***	0
003	537	***028***	15	***053***	0	***078***	0	***103***	0
004	2094	***029***	47	***054***	0	***079***	0	***104***	0
005	990	***030***	98	***055***	0	***080***	0	***105***	0
006	6955	***031***	6575	***056***	2099	***081***	684	***106***	0
007	6324	***032***	4098	***057***	2260	***082***	172	***107***	516
008	6865	***033***	5468	***058***	4936	***083***	798	***108***	3354
009	16225	***034***	2648	***059***	3784	***084***	617	***109***	0
010	9737	***035***	5290	***060***	3289	***085***	776	***110***	0
011	17465	***036***	19732	***061***	20281	***086***	10262	***111***	31478
012	19312	***037***	17349	***062***	13403	***087***	18646	***112***	21169
013	29256	***038***	24499	***063***	19771	***088***	10021	***113***	27077
014	14377	***039***	19008	***064***	24346	***089***	25881	***114***	19648
015	26914	***040***	19611	***065***	14905	***090***	8159	***115***	13774
016	72317	***041***	57640	***066***	65386	***091***	47683	***116***	46770
017	78623	***042***	81462	***067***	65756	***092***	43004	***117***	50364
018	74310	***043***	65134	***068***	78451	***093***	55730	***118***	25460
019	77122*	***044***	78139	***069***	81627	***094***	59494	***119***	66707
020	63229	***045***	66579	***070***	68242	***095***	42688	***120***	69019
021	77774	***046***	64451	***071***	90486	***096***	126048	***121***	122266
022	100484	***047***	113999	***072***	115249	***097***	114686	***122***	82456
023	135618	***048***	74323	***073***	68529	***098***	112102	***123***	75118
024	119947	***049***	110295	***074***	79006	***099***	98206	***124***	73041
025	128747	***050***	95616	***075***	98110	***100***	157296	***125***	104531

* not solved optimally

Table F.2. Optimal/best-known solutions to the 50-job problem set – SMTWT

50-JOB PROBLEM SET									
Instance	***Optimal***	***Instance***	***Optimal***	***Instance***	***Optimal***	***Instance***	***Optimal***	***Instance***	***Optimal***
001	2134	***026***	2	***051***	0	***076***	0	***101***	0
002	1996	***027***	4	***052***	0	***077***	0	***102***	0
003	2583	***028***	755	***053***	0	***078***	0	***103***	0
004	2691	***029***	99	***054***	0	***079***	0	***104***	0
005	1518	***030***	22	***055***	0	***080***	0	***105***	0
006	26276	***031***	9934	***056***	1258	***081***	816	***106***	0
007	11403	***032***	7178	***057***	3679	***082***	4879	***107***	1717
008	8499	***033***	4674	***058***	2522	***083***	973	***108***	0
009	9884	***034***	4017	***059***	3770	***084***	508	***109***	6185
010	10655	***035***	6459	***060***	5904	***085***	3780	***110***	1295
011	43504*	***036***	34892*	***061***	25212	***086***	20751	***111***	27310*
012	36378*	***037***	22739	***062***	17337	***087***	36053*	***112***	15867
013	45383	***038***	29467	***063***	30729	***088***	28268*	***113***	35106
014	51785*	***039***	49352	***064***	18082	***089***	28846	***114***	15467
015	38934	***040***	26423	***065***	25028	***090***	15451	***115***	10574
016	87902	***041***	71111	***066***	76878*	***091***	89298	***116***	35727
017	84260	***042***	90163	***067***	85413	***092***	66340	***117***	71922
018	104795	***043***	84126	***068***	92756	***093***	61060	***118***	65433
019	89299*	***044***	123893*	***069***	77930	***094***	42453	***119***	106043
020	72316	***045***	79883	***070***	74750	***095***	56522	***120***	101665
021	214546	***046***	157505	***071***	150580	***096***	177909	***121***	78315
022	150800	***047***	133289	***072***	131680	***097***	139591	***122***	119925
023	224025	***048***	191099	***073***	98494	***098***	148906	***123***	101157
024	116015	***049***	150279	***074***	135394	***099***	179264	***124***	139488
025	240179	***050***	198076	***075***	135677	***100***	120108	***125***	110392

* not solved optimally

Table F.3. Best-known solutions to the 100-job problem set (Crauwels et al., 1998)

100-JOB PROBLEM SET									
Instance	***B.K.***	***Instance***	***B.K.***	***Instance***	***B.K.***	***Instance***	***B.K.***	***Instance***	***B.K.***
001	5988	***026***	8	***051***	0	***076***	0	***101***	0
002	6170	***027***	718	***052***	0	***077***	0	***102***	0
003	4267	***028***	27	***053***	0	***078***	0	***103***	0
004	5011	***029***	480	***054***	0	***079***	0	***104***	0
005	5283	***030***	50	***055***	0	***080***	0	***105***	0
006	58258	***031***	24202	***056***	9046	***081***	1400	***106***	0
007	50972	***032***	25469	***057***	11539	***082***	317	***107***	1193
008	59434	***033***	32964	***058***	16313	***083***	1146	***108***	0
009	40978	***034***	22215	***059***	7965	***084***	136	***109***	232
010	53208	***035***	19114	***060***	19912	***085***	284	***110***	0
011	181649	***036***	108293	***061***	86793	***086***	66850	***111***	159138
012	234179	***037***	181850	***062***	87067	***087***	84229	***112***	174377
013	178840	***038***	90440	***063***	96563	***088***	55544	***113***	91171
014	157476	***039***	151701	***064***	100788	***089***	54612	***114***	168297
015	172995	***040***	129728	***065***	56510	***090***	75061	***115***	70190
016	407703	***041***	462324	***066***	243872	***091***	248699	***116***	370631
017	332804	***042***	425875	***067***	401023	***092***	311022	***117***	324437
018	544838	***043***	320537	***068***	399085	***093***	326258	***118***	246243
019	477684	***044***	360193	***069***	309232	***094***	273993	***119***	293576
020	406094	***045***	306040	***070***	222684	***095***	316870	***120***	267326
021	898925	***046***	829828	***071***	640816	***096***	495516	***121***	471214
022	556873	***047***	623356	***072***	611362	***097***	636903	***122***	570459
023	539716	***048***	748988	***073***	623429	***098***	680082	***123***	397029
024	744287	***049***	656693	***074***	584628	***099***	622464	***124***	431115
025	585306	***050***	599269	***075***	575274	***100***	449545	***125***	560754

B.K. denotes Best- Known

Table F.4. Best-known solutions to the 100-job problem set (Congram et al., 2002)

100-JOB PROBLEM SET									
Instance	***B.K.***	***Instance***	***B.K.***	***Instance***	***B.K.***	***Instance***	***B.K.***	***Instance***	***B.K.***
001	5988	***026***	8	***051***	0	***076***	0	***101***	0
002	6170	***027***	718	***052***	0	***077***	0	***102***	0
003	4267	***028***	27	***053***	0	***078***	0	***103***	0
004	5011	***029***	480	***054***	0	***079***	0	***104***	0
005	5283	***030***	50	***055***	0	***080***	0	***105***	0
006	58258	***031***	24202	***056***	9046	***081***	1400	***106***	0
007	50972	***032***	25469	***057***	11539	***082***	317	***107***	1193
008	59434	***033***	32964	***058***	16313	***083***	1146	***108***	0
009	40978	***034***	22215	***059***	7965	***084***	136	***109***	232
010	53208	***035***	19114	***060***	19912	***085***	284	***110***	0
011	181649	***036***	108293	***061***	86793	***086***	66850	***111***	159123
012	234179	***037***	181850	***062***	87067	***087***	84229	***112***	174367
013	178840	***038***	90440	***063***	96563	***088***	55544	***113***	91169
014	157476	***039***	151701	***064***	100788	***089***	54612	***114***	168266
015	172995	***040***	129728	***065***	56510	***090***	75061	***115***	70190
016	407703	***041***	462324	***066***	243872	***091***	248699	***116***	370614
017	332804	***042***	425875	***067***	401023	***092***	311022	***117***	324437
018	544838	***043***	320537	***068***	399085	***093***	326258	***118***	246237
019	477684	***044***	360193	***069***	309232	***094***	273993	***119***	293571
020	406094	***045***	306040	***070***	222684	***095***	316870	***120***	267316
021	898925	***046***	829828	***071***	640816	***096***	495516	***121***	471214
022	556873	***047***	623356	***072***	611362	***097***	636903	***122***	570459
023	539716	***048***	748988	***073***	623429	***098***	680082	***123***	397029
024	744287	***049***	656693	***074***	584628	***099***	622464	***124***	431115
025	585306	***050***	599269	***075***	575274	***100***	449545	***125***	560754

Table F.5. Min-max best-found solution for five seed experimentation (40-job) SMTWT

40-JOB PROBLEM SET								
Instance	***Min***	***Max***	***Instance***	***Min***	***Max***	***Instance***	***Min***	***Max***
001	913	913	***043***	65134	65134	***085***	776	776
002	1225	1225	***044***	78139	78139	***086***	10262	10262
003	537	537	***045***	66579	66579	***087***	18646	18706
004	2094	2094	***046***	64451	64451	***088***	10027	10027
005	990	990	***047***	113999	113999	***089***	25881	25881
006	6955	6955	***048***	74323	74323	***090***	8159	8159
007	6324	6324	***049***	110295	110295	***091***	47683	47683
008	6865	6865	***050***	95616	95616	***092***	43004	43004
009	16225	16310	***051***	0	0	***093***	55730	55730
010	9737	9737	***052***	0	0	***094***	59494	59494
011	17465	17465	***053***	0	0	***095***	42688	42688
012	19312	19312	***054***	0	0	***096***	126048	126048
013	29256	29279	***055***	0	0	***097***	114686	114686
014	14432	14432	***056***	2099	2099	***098***	112102	112102
015	26914	26914	***057***	2260	2260	***099***	98206	98206
016	72317	72317	***058***	4936	4936	***100***	157296	157296
017	78623	78623	***059***	3784	3784	***101***	0	0
018	74310	74373	***060***	3289	3289	***102***	0	0
019	77122	77432	***061***	20283	20434	***103***	0	0
020	63229	63368	***062***	13403	13403	***104***	0	0
021	77774	77774	***063***	19771	19771	***105***	0	0
022	100484	100484	***064***	24346	24498	***106***	0	0
023	135618	135618	***065***	14905	15341	***107***	516	516
024	119947	119947	***066***	65386	65386	***108***	3354	3354
025	128747	128747	***067***	65756	65756	***109***	0	0
026	108	108	***068***	78451	78451	***110***	0	0
027	64	64	***069***	81688	81688	***111***	31478	31478
028	15	15	***070***	68242	68242	***112***	21169	21169
029	47	47	***071***	90486	90486	***113***	27077	27077
030	98	98	***072***	115249	115249	***114***	19648	19678
031	6575	6575	***073***	68529	68529	***115***	13774	13774
032	4098	4099	***074***	79013	79013	***116***	46770	46929
033	5468	5468	***075***	98110	98110	***117***	50364	50384
034	2648	2648	***076***	0	0	***118***	25460	25460
035	5290	5290	***077***	0	0	***119***	66707	66707
036	19732	20095	***078***	0	0	***120***	69042	69042
037	17349	17349	***079***	0	0	***121***	122266	122266
038	24563	24630	***080***	0	0	***122***	82456	82456
039	19008	19008	***081***	684	684	***123***	75118	75379
040	19611	19611	***082***	172	172	***124***	73041	73041
041	57640	57640	***083***	798	798	***125***	104531	104531
042	81462	81462	***084***	617	617			

Table F.6. Min-max best-found solution for five seed experimentation (50-job) SMTWT

50-JOB PROBLEM SET								
Instance	***Min***	***Max***	***Instance***	***Min***	***Max***	***Instance***	***Min***	***Max***
001	2134	2134	***043***	84126	84126	***085***	3780	3796
002	1996	1996	***044***	123893	123893	***086***	20751	20828
003	2583	2583	***045***	79883	79883	***087***	36053	36053
004	2691	2691	***046***	157505	157505	***088***	28671	28671
005	1518	1518	***047***	133289	133289	***089***	28846	28849
006	26409	26509	***048***	191099	191099	***090***	15451	15547
007	11403	11403	***049***	150279	150279	***091***	89324	89474
008	8499	8610	***050***	198076	198076	***092***	66340	66340
009	9918	9918	***051***	0	0	***093***	61060	61225
010	10655	10655	***052***	0	0	***094***	42453	42564
011	43504	43504	***053***	0	0	***095***	56632	56726
012	36459	36941	***054***	0	0	***096***	177909	178115
013	45383	45383	***055***	0	0	***097***	139591	139591
014	51785	51785	***056***	1258	1258	***098***	148906	148906
015	38934	39293	***057***	3679	3693	***099***	179264	179274
016	87902	87902	***058***	2522	2564	***100***	120108	120108
017	84260	84375	***059***	3770	3770	***101***	0	0
018	104795	104795	***060***	5904	5904	***102***	0	0
019	89299	89299	***061***	25738	25738	***103***	0	0
020	72316	72316	***062***	17515	17644	***104***	0	0
021	214546	214546	***063***	30737	30737	***105***	0	0
022	150800	150800	***064***	18082	18082	***106***	0	0
023	224025	224025	***065***	25049	25049	***107***	1717	1717
024	116015	116015	***066***	76878	76878	***108***	0	0
025	240179	240179	***067***	85501	85920	***109***	6185	6185
026	2	2	***068***	92897	93446	***110***	1295	1295
027	4	4	***069***	77933	78031	***111***	27310	27389
028	755	755	***070***	74750	74932	***112***	15867	15867
029	99	99	***071***	150580	150580	***113***	35106	35106
030	22	22	***072***	131680	131681	***114***	15505	15505
031	9934	9934	***073***	98494	98494	***115***	10574	10574
032	7260	7322	***074***	135463	135463	***116***	35727	35727
033	4674	4674	***075***	135677	135677	***117***	71927	71927
034	4017	4017	***076***	0	0	***118***	65433	65433
035	6459	6459	***077***	0	0	***119***	106043	106116
036	34892	34892	***078***	0	0	***120***	101667	101672
037	22740	22783	***079***	0	0	***121***	78315	78315
038	29492	29492	***080***	0	0	***122***	119925	119925
039	49352	49352	***081***	816	816	***123***	101157	101157
040	26423	26423	***082***	4879	4879	***124***	139488	139488
041	71111	71111	***083***	973	973	***125***	110392	110392
042	90163	90334	***084***	508	508			

Table F.7. Min-max best-found solution for five seed experimentation (100-job) SMTWT

100-JOB PROBLEM SET								
Instance	***Min***	***Max***	***Instance***	***Min***	***Max***	***Instance***	***Min***	***Max***
001	5988	6066	***043***	320640	321363	***085***	284	284
002	6170	6170	***044***	360193	360193	***086***	67166	67240
003	4286	4336	***045***	306624	307754	***087***	84303	84454
004	5011	5011	***046***	829867	829899	***088***	55931	56370
005	5283	5283	***047***	623362	623441	***089***	55808	57792
006	58258	58258	***048***	749018	749053	***090***	75223	75229
007	51318	52104	***049***	656713	656715	***091***	248823	248841
008	59809	60498	***050***	599269	599375	***092***	311514	311557
009	41005	41492	***051***	0	0	***093***	326406	326535
010	53663	53923	***052***	0	0	***094***	274073	274200
011	183665	185688	***053***	0	0	***095***	317764	319142
012	236528	237477	***054***	0	0	***096***	495657	496057
013	180988	182925	***055***	0	0	***097***	636903	636903
014	157946	159314	***056***	9046	9046	***098***	680104	680194
015	174195	175378	***057***	11740	12425	***099***	622491	622633
016	407722	408283	***058***	16313	17156	***100***	449613	449613
017	333415	335722	***059***	7965	7965	***101***	0	0
018	545801	545801	***060***	19919	19919	***102***	0	0
019	479272	480333	***061***	86915	87928	***103***	0	0
020	406675	406869	***062***	87240	90176	***104***	0	0
021	898925	898925	***063***	97315	97357	***105***	0	0
022	556873	556873	***064***	102826	106471	***106***	0	0
023	539716	539716	***065***	58831	60108	***107***	1193	1193
024	744287	744339	***066***	243942	243942	***108***	0	0
025	585306	585306	***067***	401617	401617	***109***	232	232
026	8	8	***068***	399140	399158	***110***	0	0
027	718	718	***069***	309256	309338	***111***	159123	159571
028	27	27	***070***	222794	222794	***112***	176448	177522
029	480	480	***071***	640845	640845	***113***	91389	91777
030	50	50	***072***	611362	611374	***114***	168453	168488
031	24435	24773	***073***	623429	623559	***115***	70763	71301
032	25720	26726	***074***	584632	584733	***116***	370790	370840
033	33612	33740	***075***	575283	575297	***117***	324762	325487
034	22215	23250	***076***	0	0	***118***	246697	247688
035	19490	20016	***077***	0	0	***119***	293659	293666
036	108293	109910	***078***	0	0	***120***	267350	267434
037	182053	185246	***079***	0	0	***121***	471499	471828
038	90747	92576	***080***	0	0	***122***	570485	570591
039	152706	155157	***081***	1400	1400	***123***	397197	397908
040	131062	132165	***082***	317	317	***124***	431172	431208
041	462414	462571	***083***	1146	1146	***125***	560858	561644
042	425897	426207	***084***	136	136			

Appendix F.2. Best-Known Solutions to PMTT

In Table F.8 and Table F.9, the column *U.B.K.* shows the updated best-known solutions found by Anghinolfi and Paolucci (2006), column *B.K.* shows the best-known for Bilge et al. (2004) and finally last column *HCSS* stands for our best-found solutions which are visited by HCSS approach during the numerical experimentations. The values reported in the tables are multiplied by 100 for a better representation.

Table F.8. Best-known solution to 40-job problem set – PMTT

40-JOB PROBLEM SET							
2-MACHINE				4-MACHINE			
Instance	***U.B.K.****	***B.K.*****	***HCSS*****	***Instance***	***U.B.K.***	***B.K.***	***HCSS***
01	14071	14079	14079	***01***	0	0	0
02	3946	3946	3946	***02***	0	0	0
03	3335	3335	3335	***03***	0	0	0
04	10095	10095	10095	***04***	0	0	0
05	19662	19695	19703	***05***	0	0	0
06	26372	26372	26372	***06***	0	0	0
07	18565	18565	18565	***07***	914	914	914
08	37509	37513	38073	***08***	26	48	48
09	1055	1055	1055	***09***	0	0	0
10	1032	1038	1038	***10***	0	0	0
11	1726	1726	1726	***11***	0	0	0
12	8199	8199	8199	***12***	0	0	0
13	8382	8382	8528	***13***	2681	2807	2807
14	5839	5860	5860	***14***	2704	2704	2704
15	21561	21563	21615	***15***	1382	1388	1418
16	43395	43502	43418	***16***	0	0	0
17	15816	15816	16096	***17***	0	0	0
18	5866	5866	5866	***18***	0	0	0
19	27258	27258	27258	***19***	0	0	0
20	2887	2887	2887	***20***	0	0	0

* U.B.K. denotes updated best-known solution found by Anghinolfi and Paolucci

** B.K. denotes best-known solution found by Bilge et al.

*** HCSS denotes best-found solutions which are visited by HCSS approach during the numerical experimentations

Table F.9. Best-known solution to 60-job problem set – PMTT

60-JOB PROBLEM SET							
2-MACHINE				4-MACHINE			
Instance	*U.B.K.*	*B.K.*	*HCSS*	*Instance*	*U.B.K.*	*B.K.*	*HCSS*
01	14205	14205	14205	***01***	0	0	0
02	6528	6528	7059	***02***	3219	2737	4421
03	17296	17296	17296	***03***	59	155	155
04	72330	72406	74132	***04***	0	0	0
05	34568	34640	37542	***05***	2591	2591	2792
06	50138	50492	53000	***06***	364	339	364
07	26535	26660	26660	***07***	4744	4744	4773
08	8030	8042	8051	***08***	0	0	0
09	16739	16790	16790	***09***	0	0	0
10	20899	20943	22104	***10***	4560	4626	4687
11	11204	11204	11204	***11***	4329	4423	4423
12	14080	14080	14080	***12***	0	0	0
13	12806	12806	13170	***13***	0	0	0
14	6793	6874	6834	***14***	0	0	0
15	20017	20017	20221	***15***	0	0	0
16	23981	23883	23981	***16***	49	58	53
17	12222	12222	12850	***17***	0	0	0
18	38642	38948	38642	***18***	0	0	0
19	133	164	133	***19***	0	0	0
20	23511	23514	26622	***20***	0	0	0

Table F.10. Min-max best-found solution for five seed experimentation (40-job) PMTT

40-JOB PROBLEM SET					
2-MACHINE			4-MACHINE		
Instance	*Min*	*Max*	*Instance*	*Min*	*Max*
01	14079	14079	***01***	0	0
02	3946	3946	***02***	0	0
03	3335	3335	***03***	0	0
04	10095	10095	***04***	0	0
05	19703	19863	***05***	0	0
06	26372	26372	***06***	0	0
07	18565	18565	***07***	914	1189
08	38149	39465	***08***	48	48
09	1055	1055	***09***	0	0
10	1038	1038	***10***	0	0
11	1726	1726	***11***	0	0
12	8199	8199	***12***	0	0
13	8528	8888	***13***	2807	3070
14	5860	5868	***14***	2704	2898
15	21615	22655	***15***	1418	1692
16	43418	43429	***16***	0	0
17	16096	16096	***17***	0	0
18	5866	5866	***18***	0	0
19	27258	27258	***19***	0	0
20	2939	2939	***20***	0	0

Table F.11. Min-max best-found solution for five seed experimentation (60-job) PMTT

60-JOB PROBLEM SET					
2-MACHINE			4-MACHINE		
Instance	***Min***	***Max***	***Instance***	***Min***	***Max***
01	14205	14205	***01***	0	0
02	7059	7120	***02***	4421	4847
03	17296	17296	***03***	155	155
04	74132	74132	***04***	0	0
05	37542	37542	***05***	2792	2874
06	53000	54698	***06***	364	401
07	26660	26662	***07***	4773	4886
08	8051	8051	***08***	0	0
09	16790	16790	***09***	0	0
10	22104	22649	***10***	4687	4878
11	11204	11204	***11***	4423	4423
12	14080	14080	***12***	0	0
13	13170	13485	***13***	0	0
14	6834	6882	***14***	0	0
15	20221	20450	***15***	0	0
16	23981	24592	***16***	53	58
17	12850	12850	***17***	0	0
18	38642	38948	***18***	0	0
19	133	133	***19***	0	0
20	26622	26622	***20***	0	0

REFERENCES

Anghinolfi, D. and M. Paolucci, 2006, "Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach", *Computers & Operations Research*, In Press, Available online.

Azizoğlu, M., S. Kondakci, and M. Köksalan, 2003, "Single machine scheduling with maximum earliness and number tardy", *Computers & Industrial Engineering*, Vol. 45, pp. 257-268.

Bean, J.C., 1994, "Genetics and random keys for sequencing and optimization", *ORSA Journal on Computing* Vol. 6, pp. 154-160.

Beasley, J.E., 2006, *OR-Library*, http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Bilge, U., F. Kıraç, and M. Kurtulan, 2006, "An adaptive genetic algorithm for the parallel machine total tardiness problem", Research Papers, Boğaziçi University, FBE-IE-05/2006-07.

Bilge, U., F. Kıraç, M. Kurtulan, and P. Pekgün, 2004, "A tabu search algorithm for parallel machine total tardiness problem", *Computers & Operations Research*, Vol. 31, pp. 397-414.

Bilge, U., F. Kıraç, and M. Kurtulan, 2007, "A tabu search algorithm for the single machine total weighted tardiness problem", *European Journal of Operational Research*, Vol. 176, pp. 1423-1435.

Carroll, D.C., 1965, *Heuristic Sequencing of Single and Multiple Components*, Ph.D. Dissertation, Massachusetts Institute of Technology, MA.

Congram, R.K., C.N. Potts, and S.L. Van de Velde, 2002, "An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem", *Informs Journal on Computing* Vol.14 (1), pp. 52–67.

Crauwels, H.A.J., C.N. Potts and L.N. Van Wassenhose, 1998, "Local search heuristics for single machine total weighted tardiness scheduling problem", *Informs Journal on Computing*, Vol. 10, pp. 341-350.

Dessouky, M.M., 1998, "Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize maximum lateness", *Computers and Industrial Engineering*, Vol. 34, pp. 793-806.

Dorigo, M. and G. Di Caro, 1999, "The Ant Colony Optimization Meta-Heuristic", in: Corne, D., M. Dorigo and F. Glover (Eds.), *New Ideas in Optimization*, McGraw-Hill, pp. 11-32.

Du, J., and J.Y.T. Leung, 1990, "Minimizing total tardiness on one machine is NP-hard", *Mathematics of Operations Research*, Vol.15, pp. 483–495.

Elmaghraby, S.E., and S.H. Park, 1974, "Scheduling jobs on a number of identical machines", *AIEE Transactions*, Vol. 6, pp. 1-13.

Ergun, O., J.B. Orlin, and A.B. Punnen, 2002, "A survey of very large-scale neighborhood search techniques", *Discrete Applied Mathematics*, Vol.123, pp. 75-102.

Feldmann, M. and D. Bishop, 2003, "Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches", *Computers & Industrial Engineering*, Vol. 44, pp. 307–323.

França, P.M., A. Mendes and P.Moscato, 2001, "A memetic algorithm for the total tardiness single machine scheduling problem", *European Journal of Operational Research*, Vol. 132, pp. 224-242.

Garcia, C.G., D. Pérez-Brito, V. Campos and R. Martí, 2006, "Variable neighborhood search for the linear ordering problem", *Computers & Operations Research*, Vol. 33, pp. 3549-3565.

Glover, F., 1977, "Heuristics for integer programming using surrogate constraints", *Decision Sciences* Vol. 8, pp.156–166.

Glover, F. and M. Laguna, 1997, *Tabu search*, Kluwer Academic Publishers, London.

Glover, F., 1998, "A Template for Scatter Search and path relinking", in: Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (Eds.), *Artificial Evolution, Lecture Notes in Computer Science* 1363, Springer, pp. 13–54.

Glover, F., A. Løkketangen, and D.L. Woodruff, 2000, "Scatter Search to generate diverse MIP solutions", in: Laguna, M., J.L. Gonza´lez-Velarde, (Eds.), *OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, Kluwer Academic Publishers, pp. 299–317.

Glover, F., M. Laguna, and R. Martı´, 2003, "Scatter Search and path relinking: Advances and applications", in: Glover, F., Kochenberger, G. (Eds.), *Handbook of Metaheuristics*, Kluwer, pp. 1–36.

Goldberg, D.E., 1989, *Genetic algorithms in Search, Optimization and Machine Learning*, Addison-Wiley, Reading, MA.

Greistorfer, P., 2003, "A Tabu Scatter Search Metaheuristic for the Arc Routing Problem", *Computers & Industrial Engineering*, Vol. 44, pp. 249-266.

Greistorfer, P., 2004, "Experimental pool design: Input, output and combination strategies for scatter search", in: Resende, J.P., J.P. de Sousa, (Eds.), *Metaheuristics: Computer Decision-Making*, Kluwer Academic Publishers, Boston, pp. 1–18.

Hansen, P. and N. Mladenovic, 2001, "Variable neighborhood search: Principles and applications", *European Journal of Operational Research*, Vol. 130, pp. 449-467.

Hansen, P. and N. Mladenovic, 1999, "An introduction to variable neighborhood search". in: S. Voss et al. (Eds.), *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Dordrecht, pp. 433-458.

Hansen, P., N. Mladenovic and D. Uroševic, 2006, "Variable neighborhood search and local branching", *Computers & Operations Research*, Vol. 33, pp. 3034–3045.

Herrera, F., M. Lozano, and D. Molina, 2006, "Continuous scatter search: An analysis of integration of some combination methods and improvement strategies", *European Journal of Operational Research*, Vol. 169, pp. 450-476.

Jain A.S. and S. Meeran, 1999, "Deterministic job-shop scheduling: Past, present, and future", *European Journal of Operational Research*, Vol. 113, pp. 390-434.

Jones, A. and J. C. Rabelo, 1998, *Survey of Job Shop Scheduling Techniques*, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD.

Kennedy, J., R. and C. Eberhart, 2001, and Y. Shi, *Swarm intelligence*, Morgan Kaufmann Publishers, San Francisco.

Kethley, R.B. and B. Alidaee, 2002, "Single machine scheduling to minimize total weighted late work: a comparison of scheduling rules and search algorithms", *Computers & Industrial Engineering*, Vol. 43, pp. 509-528.

Kytöjoki, J., T. Nuortio, O. Bräysy and M. Gendreau, 2005, "An efficient variable neighborhood search heuristic for very large scale vehicle routing problems", *Computers & Operations Research*, In Press, Available online.

Laguna, M., J.W. Barnes, and F. Glover, 1991, "Tabu search methods for a single machine scheduling problem", *Journal of Intelligent Manufacturing* 2, pp. 63–74.

Laguna, M., and R. Martı´, 2003, *Scatter Search: Methodology and Implementations*, C. Kluwer Academic Publishers.

Laguna, M., R. Martı´ and F. Glover, 2006, "Principles of scatter search", *European Journal of Operational Research*, Vol. 169, pp. 359-372.

Lawler, E.L., 1977, "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness", *Annals of Discrete Mathematics*, Vol.1, pp. 331–342.

Lejeune, M.A., 2006, "A variable neighborhood decomposition search method for supply chain management planning problems", *European Journal of Operational Research*, Vol. 175, pp. 959-976.

Lenstra, J.K., K. Rinnooy, and A.H.G., Brucker, P., 1977, "Complexity of machine scheduling problems", *Annals of Discrete Mathematics*, Vol.1, 343–362.

Liaw, C.F., Y.K. Lin, C.Y. Cheng, and M. Chen, 2003, "Scheduling unrelated parallel machines to minimize total weighted tardiness", *Computers and Operations Research*, Vol. 30, pp. 1777-1789.

Liepins, G.E., and M.R. Hilliard, 1989, "Genetic Algorithms: Foundations and Applications", *Annals of Operations Research*, Vol. 21, pp. 31-58.

Liu, Y.H., 2006, "A hybrid scatter search for the probabilistic traveling salesman problem", *Computers & Operations Research*, In Press, Available online.

Marti, R., 2006, "Scatter Search-wellsprings and challenges", Editorial, *Europen Journal of Operational Research*, Vol. 169, pp.351-358.

MathWorks, 2006, "MATLAB®, Matrix laboratory, http://www.mathworks.com.

Matsuo, H., C.J. Suh, and R.S. Sullivan, 1989, "A controlled search simulated annealing method for the single machine weighted tardiness problem", *Annals of Operations Research* Vol. 21, pp. 85–108.

Montagne Jr., E.R., 1969, "Sequencing with time delay costs", *Industrial Engineering Research Bulletin* 5.

Morton, T.E. and D.W., Pentico, 1993, *Heuristic Scheduling Systems with Applications to production Systems and Project Management*, John Wiley, New York.

Nomura, T., and K. Shimohara, 2001, "An analysis of two-parent recombinations for real-valued chromosomes in an infinite population", *Evolutionary Computation Journal*, Vol. 9, pp. 283-308.

Nowicki, E. and C. Smutnicki, 2006, "Some aspects of scatter search in flow-shop problem", *European Journal of Operational Research*, Vol. 169, pp. 654-666.

Osman I.H. and G. Laporte, 1996, "Metaheuristics: A bibliography", *Annals of Operational Research*, Vol. 63, pp. 513-628.

Osman, I.H. and J.P. Kelly, 1996, "Meta-heuristics: An overview". in: Osman, I.H. and J.P. Kelly, *Meta-heuristics: Theory and applications*, Kluwer, Boston, pp. 1–21.

Pacheco, J. A., 2005, "A scatter search approach for the minimum sum-of-squares clustering problem", *Computers & Operations Research*, Vol. 32, pp.1325–1335.

Panwalkar, S.S., Smith, and M.L. Koulamas, C.P., 1993, "A heuristic for the single machine tardiness problem", *European Journal of Operations Research* Vol. 70, pp. 304–310.

Pinedo, M. and D. Simchi-Levi, 1996, "Heuristics Methods: Applications to Facility Layout, Routing and Scheduling". in: M. Avreil and B. Golany, (Eds.) *Mathematical Programming for Industrial Engineers*, Marcel Dekker, Inc., NY, pp. 575 -617.

Potts, C.N., and L.N. Van Wassenhove, 1991, "Single machine tardiness sequencing heuristics", *IIE Transactions* Vol. 23, pp. 346–354.

Rachamadugu, R.V., and T.E. Morton, 1982, *Myopic Heuristics for the Single Machine Weighted Tardiness Problem*, Working Paper 30-82-83, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Reeves, C.R., 1993, *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley, New York.

Reeves, C.R., and T. Yamada, 1998, "Genetic algorithms, path relinking and the flowshop sequencing problem", *Evolutionary Computation Journal*, Vol. 6, pp. 45–60.

Root, J.G., 1965, "Scheduling with deadlines and loss functions on k parallel machines", *Management Science*, Vol. 11, pp. 460-475.

Schaller, J., 2004, "Single machine scheduling with early and quadratic tardy penalties", *Computers & Industrial Engineering*, Vol. 46, pp. 511–532.

Silver E.A., 2004, "An overview of heuristic solution methods", *Journal of the Operational Research Society*, Vol. 55, pp. 936-956.

Sivrikaya-Şerifoğlu F. and G. Ulusoy, 1999, "Parallel machine scheduling with earliness and tardiness penalties", *Computers & Operations Research*, Vol. 26, pp. 773–787.

Yamashita, D.S., V.A. Armentano and M. Laguna, 2006, "Scatter search for project scheduling with resource availability cost", *European Journal of Operational Research*, Vol. 169, pp. 623-637.