DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MODELLING AND SOLVING OPTIMIZATION PROBLEMS IN UNCERTAIN/DYNAMIC ENVIRONMENTS WITH MANUFACTURING SYSTEMS EXAMPLES

by
Fehmi Burçin ÖZSOYDAN

April, 2016
İZMİR

MODELLING AND SOLVING OPTIMIZATION PROBLEMS IN UNCERTAIN/DYNAMIC ENVIRONMENTS WITH MANUFACTURING SYSTEMS EXAMPLES

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Industrial Engineering, Industrial Engineering Program

by
Fehmi Burçin ÖZSOYDAN

April, 2016
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MODELLING AND SOLVING OPTIMIZATION PROBLEMS IN UNCERTAIN/DYNAMIC ENVIRONMENTS WITH MANUFACTURING SYSTEMS EXAMPLES"** completed by **FEHMİ BURÇİN ÖZSOYDAN** under supervision of **PROF. DR. ADİL BAYKASOĞLU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof.Dr. Adil BAYKASOĞLU

Supervisor

Asoc.Prof.Dr. Ali Serdar TAŞAN

Thesis Committee Member

Prof.Dr. Gülay TOHUMOĞLU

Thesis Committee Member

Prof.Dr. Pınar Düzdar

Examining Committee Member

Prof. Dr. Hadi GÖKGEN

Examining Committee Member

Prof.Dr. Ayşe OKUR
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to Prof. Dr. Adil BAYKASOĞLU for his guidance, support and understanding. He not only served as my supervisor but also encouraged me throughout the progress of this dissertation.

Besides my supervisor, I would also like to thank to the committee members Assoc. Prof. Dr. Ali Serdar TAŞAN and Prof. Dr. Gülay TOHUMOĞLU, for their helpful and insightful advices.

I would like to introduce my thanks to Norm Fasteners Company engineers Umut İNCE, Utku KALETÜRK and Ragıp ŞİMŞEK for their support throughout the SAN-TEZ project supported by Republic of Turkey, Ministry of Science, Industry and Technology under the project number: SANTEZ 0293.STZ.2013-2.

I would like to point out my gratitude to the Scientific and Technological Research Council of Turkey (TÜBİTAK) for funding this dissertation by the scientific project under grant number: 214M240.

Finally, I am indebted most of all to my family, for their encouragements and endless supports in my whole life and to my dear wife Dr. Ebru ONRAT ÖZSOYDAN for her love, patience and smiles that always have given me strength and morale in my whole life since I met her.

Fehmi Burçin ÖZSOYDAN

MODELLING AND SOLVING OPTIMIZATION PROBLEMS IN UNCERTAIN/DYNAMIC ENVIRONMENTS WITH MANUFACTURING SYSTEMS EXAMPLES

ABSTRACT

In most of the optimization problems, the problem related data as well the problem domain is assumed to be exactly known beforehand and to remain stationary throughout optimization process. However, majority of real-life problems are dynamic in their nature due to unpredictable events. Such problems with time-varying problem domain or with changing parameters are referred as dynamic optimization problems in the related literature. The aim here is not only to find the global optimum of a single problem environment, but also to track the moving optima.

The importance of dynamic optimization for manufacturing systems has been stressed by many authors particularly in the last decade. However, most of the published work includes only synthetic and theoretical problems commonly used as benchmarks. Practical implementations of dynamic optimization concepts in real-life manufacturing systems are lacking. Therefore, this thesis fills a gap by focusing on implementations of these concepts on a variety of real-life manufacturing applications.

In this respect, first, a variety of approaches is developed and is tested on the well-known benchmarks including both continuous and combinatorial dynamic problems. Next, some of these approaches are tested on real-life manufacturing systems applications through funded scientific projects. Besides the practicability and simplicity in adaptation of these approaches to real-life systems, it is also shown that, satisfactory results can be achieved by these methods.

Keywords: Dynamic optimization problems, firefly algorithm, greedy randomized adaptive search procedure, time-varying knapsack problem, parallel machine scheduling problem, turret-indexing problem.

ÜRETİM SİSTEMLERİ ÖRNEKLERİNDE BELİRSİZ/DİNAMİK OPTİMİZASYON PROBLEMLERİNİN MODELLENMESİ VE ÇÖZÜMÜ

ÖZ

Optimizasyon problemlerinin birçoğunda, problemin tanımlı olduğu kümenin ve parametrelerinin önceden kesin olarak bilindiği ve optimizasyon süreci boyunca değişmediği varsayılmaktadır. Fakat gerçek hayat problemlerinin çoğunluğu önceden kestirilemeyen olaylar sebebiyle doğaları gereği dinamiktir. Bunun gibi zamana bağlı tanım kümesi ve değişken parametreleri olan problemler, dinamik optimizasyon problemleri olarak adlandırılmaktadır. Burada amaç, tek bir problemin eniyi çözümünü bulmanın yanı sıra, aynı zamanda, olaylara ya da zamana bağlı olarak değişen eniyi çözümleri sürekli olarak takip edebilmektir.

Dinamik optimizasyonun üretim sistemleri açısından önemi, özellikle son on yıldır araştırmacılar tarafından vurgulanmaktadır. Fakat yayımlanmış çalışmaların çoğunluğu, bilimsel yazında kıyaslama problemi olarak kullanılan teorik problemleri içermektedir. Gerçek hayat problemlerini kapsayan uygulamaların sayısı oldukça azdır. Mevcut çalışma, dinamik optimizasyon tekniklerinin üretim sistemlerindeki uygulamalarını içermektedir. Bu bakımdan bu çalışmanın ilgili yazındaki boşluğu dolduracağı ve gelecek çalışmalara temel teşkil edebileceği öngörülmektedir.

Bu bağlamda, öncelikle, çeşitli yaklaşımlar geliştirilmiş, sürekli ve kombinatorik dinamik optimizasyon problemlerini içeren problem setleri üzerinde performans testleri yapılmıştır. Ardından, ilgili yöntemlerin bir kısmı, desteklenen bilimsel projeler kapsamında, çeşitli gerçek hayat üretim sistemlerinde uygulanmıştır. Önerilen yaklaşımların gerçek hayat problemlerine uyarlanmasındaki kolaylık ve uygulanabilirliklerinin yanı sıra, geliştirilmiş olan bu yöntemler ile başarılı sonuçlar elde edilebileceği de gösterilmiştir.

Anahtar kelimeler: Dinamik optimizasyon problemleri, ateş böceği algoritması, açgözlü rassal arama, zamana bağlı sırt çantası problemi, paralel makine çizelgeleme problemi, taret indeksleme problemi.

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

1.1 General Framework and Motivation

In mathematics, engineering, computer science or in similar scientific research branches, optimization is one of the eldest topics, which has attracted a notable attention. In optimization, the widespread idea among researchers has been that the problem related data as well the problem domain is exactly known beforehand and remains stationary throughout the optimization process. Although this might be true for some specific problems, in most of the practical real-life applications, optimization problems are dynamic in their nature due to a number of uncontrollable and unpredictable events, such as rush orders, arrivals of new orders, cancellations of already accepted orders, breakdowns, changes in due dates, changes in batch sizes, fluctuating capacities of manufacturing constraints, changes in labor capacity or changes in outer world conjunctures like changes in costs or profits.

Branke & Schmeck (2003) report that "*However, almost all publications deal with optimization in static, non-changing environments, whereas many real-world problems are actually dynamic: new jobs have to be added to the schedule, machines may break down or wear out slowly, raw material is of changing quality, etc.*". In a more recent study, Nguyen, Yang & Branke (2012) indicate that *"Optimization in dynamic environments is a challenging but important task since many real-world optimization problems are changing overtime.*".

Through an ongoing process of short or medium term plans such as generating/applying schedules, labor assignments to some specific jobs, production planning according to short or medium term horizon, arrivals/cancellations of jobs, breakdowns, changes in production restrictions, changes in lot sizes and due dates are some examples for frequently encountered dynamic events in real-life manufacturing systems. Such problems, changing either by unpredictable events or by time-varying problem domain and changing parameters are referred as dynamic

optimization problems (DOPs) in the literature (Branke, 1999a; Branke, 2001a; Branke, 2001b; Jin & Branke, 2005; Morrison, 2004; Weicker, 2003a; Weicker, 2003b; Wilke, 1999). The changes in problem domain such as changes in problem definition sets, in the number of decision variables or constraints are referred as dimensional changes, whereas changes in problem parameters (time-varying parameters) such as changes in coefficients or right-hand-side values are commonly referred as non-dimensional changes (Mavrovouniotis, 2013). The aim in DOPs is not only to find the global optimum of a single problem environment, but also to track the moving optima so that the system could easily adapt to the new problem configuration.

For a changing problem, one can say that the used algorithm can simply be reinitialized with its initial configuration whenever a dynamic event occurs. Thus, as intuition suggests, each new environment can be handled as individual static problems. This might work for some special cases, where there is less dependency between two consecutive environments. However, in such a case, the decision maker has to risk losing all information gathered throughout optimization process and to waste time while re-optimizing the whole system from scratch. This obviously yields to loss of resources and time. On the other hand, generally speaking, if there is a dependency between two consecutive environments, currently found promising solutions can be revised and can be adapted to the new problem environment in a shorter time without restarting the whole search. Nevertheless, in this case, there is an apparent risk to trap in local optima. There is a fragile balance between these two main ideas. Therefore, in order to deal with these difficulties of this field, the researchers from dynamic optimization community have proposed numerous methods, which can generally be categorized as introducing diversity based, maintaining diversity based, memory based and multi-population based techniques.

The importance of dynamic optimization for manufacturing systems has always been stressed by many authors particularly in the last decade. However, most of the published work includes theoretical problems commonly used as benchmarking problems in operations research literature. In comparison with these studies, unfortunately, practical implementations of dynamic optimization concepts in real-

life manufacturing systems are lacking. Therefore, the present thesis focuses on implementations of these concepts on a variety of real-life applications including, parallel machine scheduling problem with sequence-dependent setup times and eligibility constraints, flexible manufacturing systems with tool switching problem, machining operations and cutting tool indexing in CNC machine turrets. Moreover, because most of the real-life problems are combinatorial, rather than employing evolutionary algorithms, making use of constructive and multi-start search strategies to capture the changing dynamics of the interested problem faster, is given priority in this dissertation.

1.2 Research Methodology

In the present thesis, first, some methods including, introducing diversity based and multi-population based approaches are developed and performances of these algorithms are tested on some benchmarking problems commonly used in the dynamic optimization literature. The aim here is to gain deep insights about the common concepts of dynamic optimization field so that these ideas could be generalized to other DOPs later on. It is shown in the experimental study sections of these benchmarking studies that the developed approaches in this thesis for continuous DOPs are found as competitive and promising. Unfortunately, most of these benchmarking problems have continuous problem domain and includes only non-dimensional changes so the developed methods cannot directly be used in dynamic real-life manufacturing problems .In addition, as to be clarified in details in the forthcoming sections of this thesis, handling dimensional changes like arrivals of new orders, cancellations of already accepted orders, breakdowns, etc., requires quite a different effort, because such changes directly affect the solution representations. What is more, most of the DOPs in manufacturing systems include dimensional changes.

In this context, evolutionary algorithms, which are widely used in traditional static optimization problems, have some apparent shortcomings while employing them in DOPs. Let' s assume that at time t there exist m decision variables so that a solution representation is comprised of m bits. Later in time t+1, a dynamic event is

triggered and some of the already accepted jobs are cancelled. Thus, now there is less than m decision variables, which directly affects the length and structure of the solution representation. Finally, the question is how to cross or compare these two solutions with m and less than m bits. Fortunately, constructive with multi-start search strategy emerges here. However, surprisingly, this has not been sufficiently debated in the related literature.

In this regard, a specialized benchmarking problem with both dimensional and non-dimensional changes is generated first. In order to obtain scientific demonstrations and proofs, the performance of such strategy is compared to the performances of several evolutionary and population based algorithms. The proposed strategy is later embedded into the enterprise resource planning (ERP) system of a company, through a scientific project, funded by Republic of Turkey, Ministry of Science, Industry and Technology and the Scientific and Technological Research Council of Turkey. As clarified later, the aim in this case study is to automatically generate efficient schedules for parallel heat treatment furnaces under highly dynamic operating conditions.

Differently from constructive and evolutionary search strategies, this thesis also includes a threshold accepting neighborhood based strategy. Some further modifications of this approach, along with its canonical version are tested on a flexible manufacturing system example with cutting operations, where inefficient ATC indexing and tooling policies yield to undesirable non-machining times. Depending on some unique features of the mentioned problem, several specialized methods like using a shortest path algorithm are proposed as well. According to the conducted extensive experimental study, it is shown that by making use of the proposed techniques, both static and dynamic modifications of this problem can efficiently be handled.

1.3 Outline of the Thesis

The following chapter is devoted to some fundamental concepts of DOPs. Those concepts involve, frequently used performance measures, widely known

benchmarking problems along with their attributes and some famous benchmark generators.

Chapter 3 presents a developed multi-population based algorithm particularly usable in multi-modal environments. In order to introduce diversity throughout search, chaos and several other extensions are also adopted in this strategy. The tests are conducted on a well-known benchmarking problem of this domain.

A comparison of evolutionary and population based algorithms with a constructive and multi-start search strategy is presented in Chapter 4. The aim here is to show and to utilize some unique features of constructive strategies. Tests here are performed on a combinational problem, which is another widely used benchmark problem in dynamic optimization.

Chapter 5 includes a case study. In this case study, the proposed constructive and multi-start search strategy is embedded into the ERP system of a company, which is currently operating under challenging dynamic conditions. The aim here is to generate efficient and applicable schedules for parallel heat treatment furnaces as fast as possible.

Chapter 6 involves an example of a flexible manufacturing system with tool switching problem, machining operations and cutting tool indexing problem in CNC machine turrets. A neighborhood based strategy with an integrated shortest path algorithm is used as a solution approach here. Next, some further modifications of this strategy along with its base version are tested on a dynamic extension of the introduced problem.

Finally, Chapter 7 presents a summary of this thesis and contributions. Possible future work directions are discussed in this chapter.

CHAPTER TWO
COMMON CONCEPTS ABOUT DYNAMIC OPTIMIZATION PROBLEMS

2.1 Chapter Introduction

Optimization problems have attracted a considerable attention of researchers from a wide variety of disciplines including mathematics, engineering, computer sciences, chemistry, medicine, etc. However, in majority of the studies, problem related data and problem domain are assumed to be exactly known beforehand and to remain stationary throughout optimization process.

Weicker (2003b) presents a formal definition of general static optimization problems as in the following.

$$\mathcal{X} = \{x \in \Omega \mid \forall_{x' \in \Omega} f(x) \succeq f(x')\} \tag{2.1}$$

where an optimization problem is defined by a closed search space Ω, a quality function $f : \Omega \longrightarrow \mathbb{R}$ and a relation denoted by $" \succ " \subset \{<, >\}$.

In the same dissertation (Weicker, 2003b), this definition is later extended to dynamic optimization problems as formulated in the following.

$$\mathcal{X}^{(t)} = \{x \in \Omega \mid \forall_{x' \in \Omega} f^{(t)}(x) \succeq f^{(t)}(x')\} \tag{2.2}$$

where a dynamic optimization problem is defined by the search space denoted by search Ω, a set of functions $f^{(t)} : \Omega \longrightarrow \mathbb{R}\ (t \in \mathbb{N}_0)$ and a relation denoted by $" \succ " \in \{<, >\}$. The aim here is to find the set of global optima $\mathcal{X}^{(t)} \subseteq \Omega\ (t \in \mathbb{N}_0)$.

In their survey Cruz, González & Pelta (2011), define DOP as a problem, where the objective or restrictions change over time. A more formal definition of Cruz et al. (2011) is presented in the following formulation.

$$\text{DOP} = \begin{cases} \text{optimize } f(x,t) \\ \text{s.t.} \quad x \in F(t) \subseteq S, t \in T \end{cases} \tag{2.3}$$

where $S \in R^n$ and S is the search space, t is the time, $f : S \times T \longrightarrow R$ is the objective function assigning a numerical value $(f(x,t) \in R)$ to each possible solution $(x \in S)$ at time t and $F(t)$ is the set of feasible solutions $x \in F(t) \subseteq S$ at time t.

Nguyen et al. (2012) define a DOP as "*Given a dynamic problem* f_t*, an optimization algorithm G to solve* f_t*, and a given optimization period* $\left[t^{begin}, t^{end}\right]$*,* f_t *is called a dynamic optimization problem in the period* $\left[t^{begin}, t^{end}\right]$ *if during* $\left[t^{begin}, t^{end}\right]$ *the underlying fitness landscape that G uses to represent* f_t *changes and G has to react to this change by providing new optimal solutions.*".

Numerous other general definitions for DOPs can be found in the related literature. All those definitions including real-life examples along with the formal definition given above can be summarized as in the following.

For a given optimization problem, problem domain (number of the variables or constraints) as well as all related parameters including objective function coefficients, right-hand-side values of constraints and coefficients of the variables defined in constraints might change throughout an ongoing optimization process subsequent to various unpredictable events like order arrivals, rush work orders, cancellations of already accepted orders, due date changes, changes in production capacities, breakdowns in workshops or supply chains (Branke, 1999a; Branke, 2001a; Branke, 2001b; Jin & Branke, 2005; Morrison, 2004; Weicker, 2003a; Weicker, 2003b; Wilke, 1999). Such a dynamically changing problem is referred as a dynamic optimization problem in this dissertation. In such dynamically changing problems, rather than finding the global optimum only once for the present problem environment, the motivation should be tracking the changing optima and providing a fast response to the dynamic events that change the problem environment.

Jin & Branke (2005) report that uncertainty in optimization problems and evolutionary computation can arise from several reasons, which can be categorized into four categories namely, noise in fitness functions, robustness, approximation in

fitness function and time-varying fitness functions. Although in any DOP, any of these characteristics can be observed, generally speaking, a notable number of publications consider time-varying fitness functions and parameters. Some other common characteristics of dynamism in DOPs are given next.

2.2 Characteristics of Dynamism

In their studies, Branke & Schmeck (2003) and Nguyen (2010) state several characteristics of dynamisms, encountered in DOPs. These criteria are summarized below.

Frequency of change

This characteristic is actually the frequency of dynamic events that change the current environment. In other words, it is a measurement of time, elapsed between two changes. However, time here depends on the used hardware or the employed algorithm. Therefore, for more fair comparisons, countable metrics such as the number of function evaluations or number of generations are used.

Severity of change

Severity of change is actually the strength of the change. More severe changes are expected to yield to more serious changes in the current problem environment, whereas less severe changes result in as slight changes.

Predictability of change

This criterion is related to predicting the severity, direction and time of a change. Therefore, here a pattern is sought by some particular techniques.

Cycle length / cycle accuracy

If the optimum returns to its previous same or close locations, then the average number of environmental states between two same or similar states can be measured. This is referred as cycle length. In addition, similarity between these states represents the accuracy. Such environments are commonly known as periodic, recurrent or

cyclic environments (Nguyen, 2010). Thus, in such environments, particularly memory based techniques (Chapter 3.2) can achieve promising results.

Time-linkage

"*Whether the future behavior of the problem depends on the current solution found by the algorithm or not.*" Nguyen (2010).

Constrained problem

Simply if constraints exist in the problem, then it is referred as a constrained problem.

Visibility of change

The changes might be either visible or invisible to the system. If the changes are invisible, the used algorithm should detect those changes itself.

Factors that change

Changing factors represent the changing parts of the problem such as parameters, variables or constraints.

Necessity to change representations

Some dynamic events yield to changes in sizes of solution representations due to changes in the dimension of the problem, whereas some others only yield to parameter based changes.

2.3 Performance Measures Frequently Used in DOPs

Evaluating algorithms in dynamic optimization is quite different from as in static optimization. In static optimization, commonly, the best and the average solutions found by the algorithm are reported along with the CPU usages or the number of performed function evaluations. On the other hand, in dynamic optimization field, there are many more criteria. While some of them measure the overall behavior of an

algorithm, capability of recovering from dynamic events, adaptability to dynamic events, some other focus on precision in finding promising solutions, convergence speed, etc.

Finding the best solution here alone is not a matter of the fact in comparison to static optimization. For example, an algorithm, which cannot find the optimum of a problem environment but has already converged to promising solutions, can easily outperform an algorithm, which wastes majority of the search in less promising regions, although it finds the optimum of the current environment through the end of the search. In a static optimization concept, the second algorithm can be assessed as a better algorithm because it finds the optimum; however, this might not be sufficient in dynamic optimization field. In their study, Alba & Sarasola (2010) report that, performance evaluation in DOPs, is still an important and not agreed issue. Several frequently employed metrics are presented in the following.

Branke & Schmeck (2003) present some fundamental performance measures (Equations 2.4-2.8) inspired from DeJong (1975). Let e_t denote the tth evaluation and assume that a number of T evaluations are considered here.

online performance

As formulated in Equation 2.4, it is the average of evaluations until time T. This metric uses the assumption that each evaluation requires testing real world cases (Branke & Schmeck, 2003).

$$x(T) = \frac{1}{T}\sum_{t=1}^{T} e_t \tag{2.4}$$

offline performance

In contrast to online performance, in offline performance, it is assumed that testing is performed in an artificial environment and only the best solution is transferred to the real world. This metric (Equation 2.5) represents the average of the best-found values so far at each time step.

$$x^*(T) = \frac{1}{T}\sum_{t=1}^{T} e_t^* \tag{2.5}$$

where $e_t^* = \max\{e_1, e_2, \dots, e_t\}$. However, offline performance can bring some problems for the changing environments because the best solution found so far (e_t^* until *t*) must belong to the current environment. In other words, this metric needs to consider the solutions since the last change of the environment. Therefore, it is extended as given in Equation 2.6.

$$x'(T) = \frac{1}{T}\sum_{t=1}^{T} e_t' \tag{2.6}$$

where $e_t' = \max\{e_\tau, e_{\tau+1}, \dots, e_t\}$ and τ is the last time step before *t* where a dynamic change occurs.

current error

This metric (Equation 2.7), is simply the difference between the current best solution and the optimum solution.

$$e_t = opt(t) - e_t' \tag{2.7}$$

offline error

The offline error, which is one of the most commonly used metric in this field is finally formulated in Equation 2.8. It is the average of all current errors over all evaluations.

$$e^*(T) = \frac{1}{T}\sum_{t=1}^{T} e_t \tag{2.8}$$

best-of-generation

In the same year, two identical measures, namely *collective mean fitness* and *mean best-of-generation*, are independently proposed by Morrison (2003) and Yang & Yao (2003), respectively. This measure is formulated in Equation 2.9. These two identical measures actually average the best found solutions of corresponding generations over a sufficient number of independent runs and generations.

$$\bar{F}_{BG} = \frac{1}{G} \times \sum_{i=1}^{i=G} \left(\frac{1}{N} \times \sum_{j=1}^{j=N} F_{BG_{ij}} \right) \tag{2.9}$$

where, $\bar{F}_{BG}$, G, N and $F_{BG_{ij}}$ represent the mean of best-of-generation, the number generations, the total number runs and the best solution found at generation i of the jth run, respectively.

In this measure, because best-found solutions at each generation are averaged, sometimes detecting the behavior of the used algorithm exactly might be difficult. For example, for a single run, obtaining exactly the same $\bar{F}_{BG}$ values for any two algorithms means that the $\bar{F}_{BG}$ values of the last generation of these two algorithms are same. However, one of these two algorithms might start with worse results and might improve the solution quality throughout the generations, whereas the other one might start with better solutions and worsen the performance through the end of the generations. In addition to the best-of-generation technique of Morrison (2003) and Yang & Yao (2003), Alba & Sarasola (2010) propose an integral based approach evaluating the area below the mean best of generations curve. To sum up, although it is one of the most famous measures in DOPs, one of the disadvantages of $\bar{F}_{BG}$, is that because it is not a normalized measure, it can be biased by the extreme values at any generation.

best-error-before-change

Trojanowski & Michalewicz (1999) introduce another measure, which they call *accuracy*. However, this metric is commonly known as *best-error-before-change* in more recent publications. In *best-error-before-change* (accuracy), the difference between the fitness of the best solution and the value of the global best solution just before a change is considered. As formulated in Nguyen (2010), this metric is presented in Equation 2.10.

$$E_B = \frac{1}{m} \sum_{i=1}^{m} e_B(i) \tag{2.10}$$

where $e_B(i)$ and m represent the best error found by the algorithm before the change of the ith environment and number of changes, respectively.

It is clear that the optimum solution of each of the environment is assumed to be known here. Additionally, because it is based on only the final errors of the last evaluation of any environment, this metric does not provide information about overall behavior of the algorithm.

peaks ratio & peak accuracy

Similar to Watson (1999), as formulated in Equations 2.11 and 2.12, Thomsen (2004) uses two measurement techniques, proposed particularly for multi-modal optimization.

$$peak\ ratio = \frac{number\ of\ peaks\ found}{total\ number\ of\ peaks} \tag{2.11}$$

$$peak\ accuracy = \sum_{j=1}^{\#peaks} |fitness(peak_j) - fitness(i)| \tag{2.12}$$

Peak ratio (Equation 2.11) is simply the ratio of the found peaks over the total number of peaks existing in the problem environment. Although it is quite a practical measure, the total number of the peaks is assumed to be known beforehand. Additionally, finding a peak might be a subjective assessment. Thomsen (2004) reports that, if the Euclidian distance between the centre of a peak and an individual of a population is less than 0.1, it is concluded that the related peak is found. This metric is evaluated over the fitness values in Watson (1999).

Peak accuracy (Equation 2.12) is the summation of the absolute differences between fitness values of the centers of each existing peak and the nearest individual *i* to those peaks. The locations of each existing peaks is assumed to be known also in this technique. Additionally computational effort might increase due to evaluation of Euclidean distances between each existing peaks and individuals.

relative ratio of the best value

Li et al. (2008) propose another metric (Equations 2.13-2.16), which is also used in competition on dynamic optimization in CEC'09 (Nguyen, 2010).

$$performance = \sum_{i=1}^{runs} \sum_{j=1}^{num\_change} r_{ij}/(num\_change \times runs) \quad (2.13)$$

$$r_{ij} = r_{ij}^{last}/\left(1 + \sum_{s=1}^{S} \left(1 - r_{ij}^{s}\right)/S\right) \quad (2.14)$$

$$r_{ij}^{s} = f_j\left(x_{best}(s)\right)/f_j(x^*) \quad (2.15)$$

$$r_{ij}^{last} = f\left(x_{best}(last)\right)/f_j(x^*) \quad (2.16)$$

where $S, f_j(x_{best}(s))$ and $f_j(x^*)$ represent the total number of sampling steps for each period, the best found values by the algorithm at the *s*th sampling of the *j*th change period and the global best of the *j*th change period, respectively.

optimization accuracy

Another metric, used dynamic optimization community is *optimization accuracy* (Equation 2.17), which is first introduced by Feng et al. (1998) for performance evaluations of evolutionary algorithms in stationary environments. This metric is later extended to dynamic optimization by Weicker (2002).

$$accuracy^g = \frac{best^g - min^g}{max^g - min^g} \quad (2.17)$$

In Equation 2.17, $accuracy^g, best^g, min^g \in \mathbb{R}$ and $max^g \in \mathbb{R}$ represent the accuracy of the used algorithm at generation *g*, best fitness achieved by the algorithm at generation *g*, minimum and maximum fitness values of the search space at generation *g*, respectively.

One of the remarkable advantages of this metric is, in comparison to other optimality based metrics like offline error, offline performance, collective means fitness; it is less biased by the extreme values found in the landscape of a problem. However, this metric has two drawbacks, where in the former one, it is assumed that the maximum and minimum values of the landscape at *g*th generation are known. This might be practical in synthetic benchmarking problems but in real-life problems, obtaining these values might be challenging. The latter one is simple. Denominator of Equation 2.17 can be equal to zero in some possible special cases.

2.4 Benchmarking Problems in Dynamic Optimization

There is a wide variety of benchmarking problems used in dynamic optimization community. However, most of these problems and related publications are based on synthetic test problems (Cruz et al.,2011).

One of the most commonly used benchmarking problems is the Moving Peaks Benchmark (MPB) (Branke, 1999b), where the aim is to find and to track the highest peak among from a number of peaks, whose coordinates, heights and widths change over time.

In this problem, after a predetermined time steps (frequency-a number of function evaluations or generations), the coordinates, heights and widths of the existing peaks are changed by random operations by making use of a Gaussian variable. An example illustration for some shifting peaks is depicted in Figure 2.1.

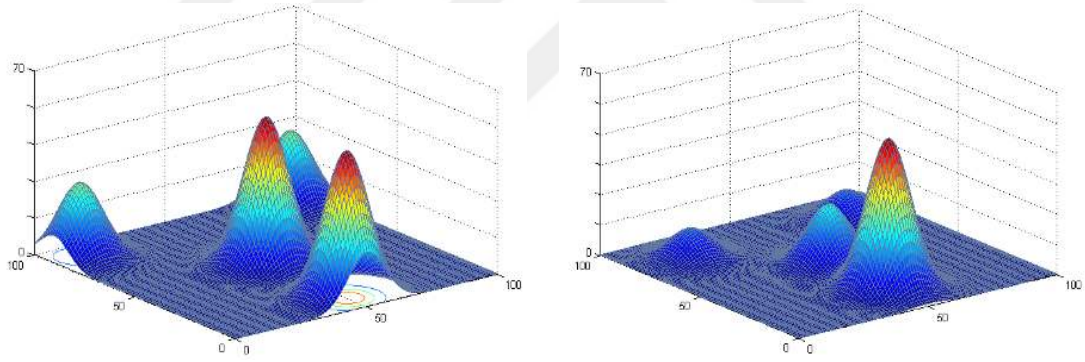


Figure 2.1 An illustration of MPB in a two-dimensional space with four peaks

A similar dynamic benchmark problem to the MPB, is proposed by Morrison & De Jong (1999) in the same year with Branke (1999b). This generator is known as DF1 and commonly referred as *field of cones* in the dynamic optimization community. As in MPB, this problem consists of peaks whose, slope, heights and coordinates change over time. In the same study, Morrison & De Jong (1999) further extend DF1 by excluding square root operation in formulation and refer this new generator as DF2. The peak tops in DF2 are smoother than the ones in DF1, which generate sharp-topped peaks.

Dynamic bit matching (dynamic match function) (Stanhope & Daida, 1998; Stanhope & Daida, 1999) is another one of the early era benchmarking problems of dynamic optimization. In this problem, the aim is to achieve a string, whose bits are similar to another target string of binary variables. Similarity contributes to the fitness, whereas, dissimilarity here means worse fitness values.

Yang (2003) uses three binary problems similar to dynamic bit matching problems, namely one-max (Ackley, 1987), royal road (Mitchell, 1992) and deceptive functions (Whitley, 1991). In one-max problem, the aim is simply to maximize the number of ones in solution strings. The optimum of this problem is achieved if all bits of a solution are represented by ones. In royal road problem of Mitchell (1992), there are 8 contiguous blocks each with 8 bits. Any of 8 blocks contributes to the objective function only if all bits of it are represented by ones. Finally, deceptive functions are a set of functions, whose combinations might produce worse solutions although they are individually promising. For example in this problem, a combination of a string (0,0,1) whose fitness is equal to 15, with another string (0,1,0) whose fitness equals to 20, can generate a new solution (0,1,1) with fitness is equal to 0.

Yang (2003) proposes an XOR operator to generate dynamic extensions for binary problems. First, a binary template (mask) T with a length, equal to the length of the chromosomes in population is generated either randomly or by some specific rules. Then, when a dynamic event is triggered, each solution x in population performs the operation of $x \oplus T$, where $\oplus$ is the bit-wise-exclusive-or (XOR) operator (i.e. $1 \oplus 1 = 0,\ 1 \oplus 0 = 1,\ 0 \oplus 0 = 0$). This operator is later used in Yang & Yao (2005).

Another benchmarking environment is referred as moving parabola (Eberhart & Shi, 2001; Hu & Eberhart, 2002), where an extension of a sphere function with a dynamic parameter referred as *offset* parameter is used.

Dynamic extensions of famous mathematical functions like Dynamic Ackley Function (Schönemann, 2004; Schönemann, 2007), Dynamic Rastrigin Function (Schönemann, 2004; Schönemann, 2007; Shi & Eberhart, 2001), Dynamic

Rosenbrock Function (Shi & Eberhart, 2001) and Dynamic Sphere (Arnold & Beyer 2002; Arnold & Beyer, 2006; Schönemann, 2004; Schönemann, 2007) are frequently adopted in dynamic optimization field. Additionally compositions of some of these functions are also used in a competition on dynamic optimization in CEC' 09 (Li et al., 2008).

As one can see from above, except the binary ones, all testing problems use a continuous problem space. However, most of the real-life problems are combinatorial in their nature. In a dissertation, Abdunnaser (2006) underlines that "*The last fifteen years have seen a growing interest in the use of evolutionary algorithms to solve dynamic optimization problems. A substantial portion of existing research targets continuous optimization problems, while a much lesser portion is directed to combinatorial problems even though many real-world problems are both discrete and time-dependent.*"

One of the most commonly used benchmarking problem as a dynamic combinatorial optimization problem (DCOP) is the dynamic extension a well-known problem, namely multi-dimensional knapsack problem, which is widely used as a benchmarking environment in stationary optimization. In dynamic multi-dimensional knapsack problem (dMKP) (Branke, Orbayı & Uyar, 2006), the profits and weights of the items as well the capacity dimensions are changed periodically. The aim here is to obtain and not lose the most profitable combination of items that does not violate constraints.

The popularity of this problem comes from the easiness in application. What is more, this problem has a wide variety of real-life extensions clarified later in this dissertation. Although its application might relatively be more difficult in comparison to dMKP, another most commonly used benchmarking problem is the dynamic extension of the vehicle routing problem (DVRP) (Mavrovouniotis, 2013; Montemanni, Gambardella, Rizzoli & Donati, 2003). In DVRP real-life conditions are reflected to the simulated problem. In order to solve some special extensions of DVRP, de-centralized optimization schemes (Baykasoğlu & Durmuşoğlu, 2011; Baykasoğlu & Durmuşoğlu, 2014) are commonly employed.

Comprehensive surveys of all other dynamic optimization related issues can be found in Abdunnaser (2006), Alba, Nakib, & Siarry (2013), Boria & Paschos (2011), Branke (1999a), Branke (2001a; 2001b), Cruz et al. (2011), Jin & Branke (2005), Li et al. (2008), Liekens (2005), Mavrovouniotis (2013), Morrison (2003), Morrision (2004), Nguyen (2010), Nguyen et al. (2012), Rohlfshagen & Yao (2008), Rohlfshagen & Yao (2009), Rohlfshagen & Yao (2011), Weicker (2002), Weicker (2003a), Weicker (2003b), Wilke (1999) and Yang, Jiang & Nguyen (2013).

2.5 Chapter Conclusion

This chapter presents some common concepts in dynamic optimization field. In this respect, first, some definitions adopted from the literature are given place here. Next, the difficulties in evaluating the performances of the used algorithms in this field are discussed. Widely used performance metrics are presented throughout this discussion. Afterwards, benchmarking problems, frequently used by the researchers from this community are introduced and it is stressed that most these benchmarks are synthetic problems. Finally, some more realistic test instances are presented at the end of this chapter.

CHAPTER THREE
A MULTI-POPULATION BASED APPROACH FOR CONTINUOUS MULTIMODAL DYNAMIC OPTIMIZATION PROBLEMS

This chapter is devoted to understanding the fundamental concepts of dynamic optimization. First, a relatively recent bio-inspired metaheuristic is extended as a multi-population based algorithm. Next, the developed approach is further enhanced by employing chaos and similar techniques. Finally, the performance of the developed approach is tested on a synthetic problem, which is commonly used as a benchmarking environment in this field.

3.1 Chapter Introduction

In most of the DOPs, the new problem environment evolves from its current form. Therefore, subsequent to occurrence of dynamic events, using and modifying the gathered information throughout the search might be a more efficient strategy in comparison to restarting the whole optimization process from scratch. In this respect, even keeping trajectories of previously found sub-optimal solutions is necessary since one of them might become the global optimum in forthcoming generations. Thus, it can be concluded that, monitoring divers parts of the problem space and keeping track of those parts are vital for a faster reaction to dynamic events. For this reason, one of the common approaches employed by researchers is to divide the whole population into smaller sized sub-populations, scattered around the search space.

However, employing multi-populations alone might not be sufficient, unless loss of diversity problem, which is another critical issue in DOPs is overcome. Due to intrinsic features of evolutionary algorithms, through generations, similarity amongst individuals of a population might increase up to undesirable levels. Accordingly, such a population loses the capability of escaping local optima or even generating new solutions dissimilar to their parents. In other words, although partitioning the whole population into sub-populations might prevent algorithm from converging to a single promising region, the loss of diversity problem is still present. In order to

overcome this problem, there exist several proposed methods, which are commonly referred as introducing/maintaining diversity techniques in literature.

In this chapter, a multi-population firefly algorithm (FA), (X.-S. Yang, 2008; X.-S. Yang, 2009) is proposed as a new solution methodology to deal with such challenges of DOPs. The proposed approach introduces several other contributions such as, employing Lévy flights and making use of several chaotic maps along with their probabilistic use (piecewise chaos), to achieve a higher level of diversity. Additionally, a standstill mode, where possibly non-contributing evaluations are postponed for a period, is employed as an auxiliary procedure. In the proposed approach, only the best sub-population is allowed to survive on the overcrowded peaks; this strategy can also be considered as another CPU related improvement. Finally, a local search procedure for the best sub-population is utilized in order to further improve the solution quality. The proposed approach is referred as multi-population chaotic firefly algorithm (mpcFA) throughout the rest of this chapter. Performance tests of the mpcFA are conducted on the well-known Moving Peaks Benchmark and results are compared to other algorithms from the literature.

The rest of this chapter is organized as follows: literature review is given in Section 3.2, the details of mpcFA and experimental results are presented in Section 3.3 and Section 3.4, respectively. Finally, chapter concluding remarks are given in Section 3.5.

3.2 Related Literature Review

There exists a variety of dynamic optimization techniques in the literature (Cruz et al., 2011; Jin & Branke, 2005; Jordehi, 2014). Using metaheuristics is one of the most widespread approaches for modeling and solving DOPs. However, as mentioned in the previous section, several intelligent strategies need to be employed in order to improve the success of metaheuristics for solving DOPs. These strategies are presented in the following.

3.2.1 Introducing Diversity Based Methods

Introducing diversity techniques play an essential role to avoid diversity related problems. One of the easiest ways is to re-initialize the algorithm with its initial configuration, and to handle the changing problems separately as static problems (Krishnakumar, 1990). This strategy might provide a quick solution to the loss of diversity problem particularly for evolutionary algorithms to some extent. However, re-initializing will result in loss of history and knowledge acquired throughout generations. In some studies (Baykasoğlu & Ozsoydan, 2014; Hu & Eberhart, 2002; Woldesenbet & Yen, 2009; Yu & Suganthan, 2009) a partial re-initialization is proposed to overcome this incident. Another famous strategy of this category is hyper-mutation, proposed by Cobb (1990). In this technique, mutation rate is increased after detecting a dynamic event. Nanayakkara, Watanabe & Izumi (1999) use chaotic maps within mutation operator. Vavak, Jukes & Fogarty (1998) further improve the studies of Cobb (1990) by using adaptive step sizes in mutation. Rossi, Barrientos & Cerro (2007) propose adaptive mutation operators for tracking changing optima. jDE and SaDE are used as self-adaptive parameters based approach in Brest, Zumer & Maucec (2006) and Qin, Huang & Suganthan (2009), respectively. In a recent study, Lepagnot, Nakib, Oulhadj & Siarry (2013) present several local search variants for continuous DOPs. Other adaptive parameters based techniques are presented in Yang & Yao (2005), Cadenas et al. (2011) and Karimi, Nobahari & Pourtakdoust (2012).

3.2.2 Maintaining Diversity Based Methods

As an alternative to introducing diversity based approaches, maintaining diversity techniques keep diversity level above a threshold throughout the search. Random immigrants (Grefenstette, 1992; Tinós & Yang, 2007; Yang, 2008) method is an example for this category. In this strategy, some random solutions are appended to the population at each generation. A similar technique is followed by Blackwell & Branke (2004), where quantum particles are employed. Inspired from the repulsion theory, charged PSO is proposed by Blackwell (2003). Li (2004) introduce another extension of PSO, where the global best particle is replaced with the neighborhood

best. Cellular PSO (Hashemi & Meybodi, 2009a) and its further improvement (Hashemi & Meybodi, 2009b), which utilize local information exchange and cellular automata distribution, are some examples of this category. A proximity structure and hierarchal topology are employed in some other modifications of PSO namely, FGPSO (Kennedy & Mendes, 2002) and HPSO (Janson & Middendorf, 2004).

3.2.3 Memory Based Methods

Another commonly employed strategy for solving DOPs is to record the previously found promising solutions and to use them in the forthcoming problem environments. An extensive survey of this category can be found in Branke (1999a), Branke (2001a), Branke (2001b) and Jin & Branke (2005). The use of memory can be classified into two sub-categories, where in the former one, memory is utilized implicitly via associative procedures and in the latter one, the past information is explicitly used.

3.2.3.1 Implicit Memory Based Methods

The use of implicit memory is introduced by Goldberg & Smith (1987). In this method, redundant solution representation with diploid genomes is utilized. Thus, the memory can implicitly be integrated into solution representations of any metaheuristic algorithm. A decade later, Hadad & Eick (1997) propose an extension of the same idea. Incorporations of these approaches in DOPs are presented in Uyar & Harmanci (2005) and Yang (2006b)

3.2.3.2 Explicit Memory Based Methods

In this method, previously found good solutions are used occasionally either to introduce diversity or to predict the promising regions of upcoming problem environments. Karaman, Uyar & Eryiğit, (2005) report a memory indexing evolutionary algorithm, with some concepts inspired from hyper-mutation and distribution estimation. Yang (2005; 2006a) and Yang, Cheng & Wang (2010) propose to generate new individuals so as to increase the level of diversity by making use of memory. In some other studies, (Pelta, Cruz & Verdegay, 2009; Richter, 2010;

Richter & Yang, 2008; Richter & Yang, 2009; Simões & Costa, 2008; Yang & Xin, 2008) the probabilities of dynamic events, landscapes of promising regions and feasible regions are derived from memory records to be utilized in upcoming environments.

3.2.4 Multi-population Based Methods

Multi-population based methods have been one of the most commonly used strategies in DOPs, particularly if the problem is multi-modal. In such problems, each peak in the landscape is actually a candidate for the optimum solution. Trying to solve DOPs by using multi-populations rather than using a single population, appears to be the most effective strategy.

The idea of using multi-populations is first introduced by Goldberg & Richardson (1987), where a niching procedure is employed. A decade later, Petrowski (1996) extends this idea by utilizing the sharing concepts within the same niches. This motivation is later adopted in some other studies (Bird & Li, 2006; Brits, Engelbrecht & van Den Bergh, 2002). Oppacher & Wineberg (1999) propose another metaheuristic approach, namely shifting balance genetic algorithm (SBGA) by using a core population for searching and by using smaller sized sub-populations for intensification of the found promising regions (Wineberg & Chen, 2004). As another concept of this category, using multiple nations, where a nation is comprised of government, population and a policy, is proposed by Ursem (1999; 2000). In that study, government and policy are defined as some of the higher quality solutions and the best solution, respectively. Inspired from forking GA of Tsutsui, Fujimoto, & Ghosh (1997), an interesting work, namely self organizing scouts (SOS) is proposed by Branke, Kaussler, Smidt, & Schmeck (2000). In SOS, search is initialized as in traditional genetic algorithms; however, after achieving a satisfactory level of convergence, a part of the population, which forms up a sub-population, is separated from the whole population. This idea of using core and sub-populations as parent and child swarms is borrowed in later studies (Cheng & Yang, 2010; Kamosi, Hashemi, & Meybodi, 2010a; Kamosi, Hashemi & Meybodi, 2010b; Kordestani, Rezvanian, & Meybodi, 2014; Li & Yang, 2008; Lung & Dumitrescu, 2007). Li, Branke &

Blackwell (2006) and Parrott & Li (2006) further improve another novel study known as speciation based PSO (SPSO) (Li, 2004; Parrott & Li, 2004). In that approach a sub-population is comprised of the particles within a distance from the best particle (seed) of the corresponding species. One year later, Bird & Li (2007), propose a new approach (rSPSO) by utilizing evolutionary and statistical methods. Using charged swarms is introduced by Blackwell & Bentley (2002). In this technique, the atomic swarm, where some of the particles are positioned in the nucleus, and the rest of the charged particles generate repulsive forces between each pair of particle, is used as a metaphor. Similar idea along with several enhancements is adopted in Blackwell & Branke (2006). In that paper, two distinct algorithms, namely, mQSO and mCPSO are proposed, where in the former one quantum and in the latter one, charged particles are used. Du & Li (2008) introduce an algorithm, which they call as multi-strategy ensemble PSO (MEPSO). The main idea here is to split the whole population into two sub-swarms, referred to as part I and part II in regard to a priori user defined proportion. Li & Yang (2009) and Yang & Li (2010) propose making use of clustering techniques (CPSO) while generating sub-populations. CPSO is further improved in Blackwell, Branke & Li (2008). Hybridizing PSO with evolutionary algorithms is proposed by Lung & Dumitrescu (2010). A few years later, a similar hybrid strategy of PSO with DE is proposed in Xiao & Zuo (2012). Some studies (Kamosi et al., 2010a; Yazdani, Nasiri, Sepas-Moghaddam & Meybodi, 2013) utilize a hibernating or sleeping-awaking method, where search of a sub-swarm is terminated. In their work, Rezazadeh, Meybodi & Naebi (2011) employ adaptive parameters and a local search procedure for the best particle of the swarm. A similar work including local search is presented in Sepas-Moghaddam, Yazdani, Arabshahi & Dehshibi (2012). Li, Yang & Yang (2014) present another self-adaptive approach for DOPs. The studies of Mendes & Mohais (2005) and du Plessis & Engelbrecht (2012) are further improve in du Plessis & Engelbrecht (2013), where a multi-population differential evolution (DE) along with several extensions is proposed. In the same year, Geshlag & Sheykhzadeh (2013) propose a learning automata strategy for PSO. Some recent studies (Daas & Batouche, 2014; Nasiri & Meybodi, 2012; Yazdani, Akbarzadeh-Totonchi, Nasiri & Meybodi, 2012; Yazdani, Nasiri, Sepas-Moghaddam, Meybodi & Akbarzadeh-

Totonchi, 2014) report multi-population based bio-inspired algorithms. Using cellular based approach, where whole search space is partitioned into smaller scaled sub-regions is another commonly used approach (Nabizadeh, Rezvanian & Meybodi, 2012a; Nabizadeh, Rezvanian & Meybodi, 2012b; Noroozi, Hashemi & Meybodi, 2011; Sharifi, Noroozi, Bashiri, Hashemi & Meybodi, 2012) of this category. Some applications of these studies are presented in Gonçalves & Resende (2012) and Toledo, Oliveira & Morelato França (2013).

3.3 Multi-population Chaotic Firefly Algorithm with Lévy Flights (mpcFA)

This section is devoted to the proposed solution methodology, which extends the canonical FA as a multi-population based approach. First, canonical FA is presented in the following.

3.3.1 Firefly Algorithm

FA is a population based algorithm, which simulates the flashing and communication behavior of fireflies (X.-S. Yang, 2008; X.-S. Yang, 2009). A comprehensive survey, including applications of this optimizer for DOPs, is presented in Fister, Fister Jr, Yang & Brest (2013) and Fister, Yang, Brest & Fister Jr. (2013).

The brightness of a firefly corresponds to the fitness of a solution. Brighter firefly attracts other fireflies. As the distance between fireflies increases, the attractiveness decreases exponentially due to the light absorption of the medium.

FA has three assumptions (X.-S. Yang, 2008; X.-S. Yang, 2009):

i. All fireflies are unisexual and every firefly attracts/gets attracted to every other firefly.

ii. The attractiveness of a firefly is directly proportional to the brightness of the firefly.

iii. They move randomly if they do not find a more attractive firefly in adjacent regions.

The distance between two fireflies (p and q) can simply be evaluated as the Euclidean distance as stated below in Equation 3.1, where n is the dimension of the problem, x_{pj} is the jth dimension of the pth firefly.

$$r_{pq} = \sqrt{\sum_{j=1}^{n} \left(x_{pj} - x_{qj}\right)^2} \tag{3.1}$$

The attractiveness of a firefly at a distanceγ_r is formulated as given in Equation 3.2, where γ is the light absorption coefficient, m is a number modifying the distance metric and $\beta_{(0)}$ is the attractiveness at zero distance.

$$\beta_{(r)} = \beta_{(0)} e^{-\gamma_r{}^m} m \geq 1 \tag{3.2}$$

Finally, movement of a firefly p is attracted to another more attractive (brighter) firefly q, is formulated in Equation 3.3, where $\alpha \in [0, 1]$ is a user supplied scaling factor of randomness and $\psi \in [0, 1]$ is a random number. A pseudo code for FA is given in Figure 3.1.

$$x_{pj} = x_{pj} + \beta_{(0)} e^{-\gamma r^2} \left(x_{qj} - x_{pj}\right) + \alpha(\psi - 0.5) \tag{3.3}$$

```
Generate initial population of fireflies
Light intensity L_p denotesthe objective value of pth firefly
while termination criterion not met
    for p=1:pop_size
        for q=1: pop_size
            if (L_q > L_p)
                move firefly p towards q in all dimensions
            end if
            Attractiveness varies with the distance r via e^(-γr^2)
            Evaluate new solution and update new light intensity L_p
        end for
    end for
    Rank the fireflies and find the current best, update the global best if required
end while
```

Figure 3.1 A pseudo code for canonical Firefly Algorithm

As the distance between fireflies increases, $\beta_{(0)}e^{-\gamma r^2}$ term quickly approaches to zero which might cause fireflies getting trapped in their positions throughout subsequent generations. In order to avoid from this incident, γ can be fixed to very small values, however, obtaining an appropriate level for γ is not an easy task. In order to overcome this shortcoming, Baykasoğlu & Ozsoydan (2014) propose a simpler move function which is given in Equation 3.4.

$$x_{pj} = x_{pj} + \beta_{(0)}\left({}^{1}/_{(\Omega + r)}\right)\left(x_{qj} - x_{pj}\right) + \alpha(\psi - 0.5) \tag{3.4}$$

where Ω is a very small number like 0.000001 to prevent division by zero. All movements of fireflies in this chapter are performed via Equation 3.4.

3.3.2 Extending FA as a Multi-population Based Strategy

In most of the previous approaches, the whole population is explicitly divided into a predefined number of sub-populations without utilizing any information about the landscape. In those works, algorithm is initialized with an arbitrary number of sub-swarms. If the number of initialized sub-swarms is greater than the number of existing peaks, this means that unnecessary evaluations are performed, which increases the CPU usage. On the other hand, if the number of peaks exceeds the number of sub-swarms, the algorithm might not be capable of detecting and covering the highest peak. In other words, rather than assigning the number of sub-swarms arbitrarily, allowing the whole algorithm to generate sub-populations one by one subsequent to detection of a peak is a more efficient technique. In this respect, the idea presented in Yazdani et al. (2013) is borrowed here.

3.3.2.1 Detecting Peaks and Generation of Colonies

In the proposed approach, first, the algorithm is initialized with a single population. The aim of this population is to detect the existing peaks in the landscape. Therefore, this population is referred as *explorer*.

Throughout the optimization process, the explorer population performs evaluations, searches on the landscape and records the best-found solution to its

history (*explorer archive*) at each generation unless a peak is detected. The decision of detecting a peak is made after a simple check on the already available history of the explorer. At the beginning of each generation, the currently best solution of the explorer is compared to the best solution found a predefined number of generations (*exp_lookback*) before. If the difference between the fitness values of these two solutions is less than a user-supplied threshold (*exp_conv*), it is concluded that the explorer population has detected a peak. Subsequent to detecting a peak, a sub-population (*colony*) is generated on the same peak. This new sub-population is exactly a copy of the explorer population. In other words, the explorer population duplicates itself on the currently found coordinates and colonizes that peak. After colonizing a peak, the explorer is re-initialized from scratch to search for possibly undetected peaks. While exploration is permanently conducted by the explorer population, colonies are tasked with exploiting the promising regions, where they are generated. Interaction between colonies is not allowed in mpcFA. In the best scenario of this procedure, each peak is covered by exactly one colony (only one more population (the explorer) exists in the landscape), which can be considered as an advantage, in terms of avoiding non-contributing evaluations.

The main difference between the explorer and colonies is the movement speed (step size) of their individuals. Because the explorer population only explores the search space and colonies perform more intensified search, the individuals of the explorer should move faster than the individuals of the colonies. This ability is satisfied by tuning the parameter $\beta_{(0)}$. Lower levels of $\beta_{(0)}$ allows a better exploitation while higher values result in faster movements of fireflies. Therefore, in mpcFA, $\beta_{(0)}$ for explorer is referred as $\beta_{(0)exp}$, while it is denoted by $\beta_{(0)col}$ for colonies. On one hand, if $\beta_{(0)exp}$ is chosen as too high, explorer would possibly miss an undetected peak. On the other hand, if it is chosen as too low, detection of uncovered peaks requires more time. It is also valid for $\beta_{(0)col}$. If it is fixed to too low values, the performance of reacting to dynamic events and tracking the moving optima might decrease. Conversely, if it is chosen as too high, this time, exploitation performance might decrease severely. In any case, it is obvious that $\beta_{(0)exp}$ should be chosen as greater than $\beta_{(0)col}$.

3.3.2.2 Exclusion

Occasionally, some unnecessary colony generations yield to overcrowded peaks, where multiple colonies are generated on the same peak. Additionally, explorer might converge to an already detected peak. In such cases, some of the existing colonies should be eliminated. This elimination is referred as *exclusion* is in the proposed approach. In mpcFA, two types of exclusion procedures are employed between the explorer and the colonies and between only colonies.

In some cases, the explorer population might converge to an already detected and colonize that peak again. In order to detect such cases, a distance-based method is used here. For this purpose, subsequent to detection of a peak by the explorer, Euclidean distances between best solutions of all existing colonies and the best solution of the explorer population are evaluated. If any of these values, is less than a predetermined value, referred as exclusion radius (r_{excl}), it is concluded that the explorer population has converged to an already colonized region (Blackwell & Branke, 2006). In such a case, the best solution of the nearest colony and the best solution of the explorer population are compared. If the best solution of the colony outperforms the best solution of the explorer, explorer is simply re-initialized. Otherwise, existing colony is excluded, a new colony is generated in place of the excluded colony and the explorer is re-initialized again. It should be noted that in both cases whenever the explorer is re-initialized, *explorer archive* is re-initialized as well.

In the best scenario, the number of generated colonies should be equal to the number of existing peaks in the landscape. The exclusion procedure mentioned above might work efficiently to some extent; however, it might also occasionally allow colonies to overlap on the same peak. In other words, some of the peaks might become overcrowded over time, whereas the rest of the peaks might remain undetected. For this reason, in order to exclude overlapping colonies covering the same peaks (exclusion between colonies), the exclusion procedure of Blackwell & Branke (2006) is adopted here again.

Similar to exclusion between the explorer and the colonies, in this procedure, Euclidean distances between the positions of the best solutions of each existing colonies are evaluated. If the distance for any pair of colonies is less than r_{excl}, it is concluded that those colonies overlap. In such a case, the colony with the best fitness is allowed to survive.

It is obvious that, evaluating Euclidian distances between all existing colonies at each generation might consume CPU. Particularly in larger scaled instances with a greater number of peaks, algorithm might suffer from these evaluations. Therefore, in the proposed approach, this procedure is performed periodically. The mentioned period is determined by employing another user supplied parameter named as *exclusion frequency* (*excl_freq*). Simply, in every *excl_freq* generations, this procedure is executed.

It should be noted that, only exclusion between colonies is performed periodically. Additionally, if the moment of this exclusion intersects with the moment of dynamic event, as an exception, periodic exclusion is not executed until *excl_freq* generations are performed subsequent to dynamic change of the environment.

3.3.2.3 Standstill Mode

In mpcFA, the colonies, which do not further improve their solution quality, are frozen until a dynamic event is detected. This period is referred as *standstill mode* (Kamosi et al., 2010a; Yazdani et al., 2013). The aim here is to eliminate unnecessary evaluations. In this mode, the frozen colonies exist in the problem space, however all actions such as movements of fireflies and objective evaluations are not performed until a dynamic event is detected.

In any approximation algorithm, it is clear that a faster convergence is expected at the initial stages of the search. This is because the gap between the initial solution and the optimum is at the highest level at those stages. Through the end of the search, it is usually observed that, convergence of the algorithm either gradually diminishes or becomes stationary. If an acceptable gap between the best-found solution and the

optimum solution is achieved, further evaluations might be terminated by the initiative of the decision maker. However, it is difficult to come to a decision for termination, because the optimum is not known priory.

In this context, a similar procedure to convergence test of the explorer is followed here with only difference that a more precise convergence threshold (*col_conv*) is used here. For this purpose, each of the existing colonies keep their own *colony archives*, where at each generation, the best solution found by each colony are recorded. Next, at the beginning of each generation, the currently best solution of each existing colonies are compared with the *col_lookback* previous solutions in their individual colony archives. For any colony, if the difference between these solutions is less than *col_conv*, it is concluded that the corresponding colony has converged and such colonies are kept in standstill mode until a dynamic event is detected. It is clear that to be able to perform this check for any colony; at least *col_lookback* generations must be performed by the corresponding colony. Additionally, if a dynamic event is detected, this check is postponed for another *col_lookback* generations beginning from the generation of the new environment. In this meantime, existing colonies are given chance to relocate their positions. Otherwise, there is a probability that this comparison might erroneously be performed between the best solution of the present environment and the best solution found in the previous environment. Clearly, such a comparison yields to failure.

3.3.2.4 Tracking Trajectories of the Moving Peaks

As mentioned previously, diversity is either maintained or explicitly introduced subsequent to detection of a dynamic event. The aim in both approaches is to perform a fast reaction to dynamic changes. A commonly used approach is to employ quantum particles (Blackwell & Branke, 2004). Those scattered particles perform random search within a distance from center of a sub-swarm, where the center is the best particle found by that sub-swarm. Thus, a faster detection of the new coordinates of moving peaks becomes possible.

In mpcFA, a new relocation method referred as *partial relocation*, is proposed. Let's assume that a peak is detected by the explorer and a colony is generated on that

peak. Again let's assume that the shift length, which is the length that the coordinates of a peak is shifted, is known beforehand, and it is equal to *r*. It means that if a change occurs, the colonized peak will shift to a new position, which is probably within a circle of radius *r*. If individuals of the colony covering that peak are relocated within the boundaries of this circle, consequently, finding the new position of the shifting peak in shorter evaluations becomes possible. However, scattering those individuals in this circle is another issue, which must be taken into consideration carefully in order to prevent overlapping of the relocated fireflies.

In the proposed method, the radius *r* is divided into three equal segments in order to obtain three concentric circles namely; *inner circle*, *middle circle* and *far circle* with radii *r*/3, 2*r*/3 and *r*, respectively (Figure 3.2). As it can be seen from Figure 3.2, all those circles originate from the same center, which is indeed the coordinates of the highest point of an arbitrary peak (denoted by "+" in Figure 3.2) just before the change. Let the current problem space be a two dimensional space, represented by *x*-axis and *y*-axis as in this figure. Thus, the area of the inner, middle and far circles are evaluated as $\frac{1}{9}\pi r^2$, $\frac{4}{9}\pi r^2$ and πr^2, respectively. Finally, the dotted (a1), vertically shaded (a2) and squared (a3) areas in Figure 3.2 can be obtained as $\frac{1}{9}\pi r^2$, $\frac{3}{9}\pi r^2$ and $\frac{5}{9}\pi r^2$, respectively. These values can be used as proportions while relocating the individuals of the same colony. In this context, a greater number of individuals can be placed to a larger area (i.e. a3), whereas a smaller area (i.e. a1) is adequate for a fewer number of individuals. The proportions of each region a1, a2 and a3 are evaluated by dividing their areas to the area of the whole circle with radius *r*. Thus, it is concluded that, $\frac{1}{9}\pi r^2/\pi r^2 \cong 0.11$, $\frac{3}{9}\pi r^2/\pi r^2 \cong 0.33$ and $\frac{5}{9}\pi r^2/\pi r^2 \cong 0.56$ of a colony should be relocated in regions a1, a2 and a3, respectively. As a result, possible overlapping and overcrowding on a specific region during relocation process is avoided to some extent. Partial relocation procedure is presented in pseudo codes given in Figure 3.3.

As it can be seen from Figure 3.3, first, a random number $rand_1 \in [0,1]$ is generated to decide where to relocate an individual. If it falls into a1, simply a random variable $rand_2 \in [-1/3r, 1/3r]$ is generated for each of the dimensions

separately. Next, in order to find new locations, those random variables are added to the corresponding dimensions of the best individual of that colony. The same procedure is followed for the regions a2 and a3 with only difference that another random number $rand_3 \in [0,1]$ is required to decide whether the random variable $rand_2$ should be positive or negative. Because, for a2, $rand_2$ cannot be directly defined in interval $[-2/3r, 2/3r]$ as in a1. Otherwise, the inner circle is covered again and possibly some of the individuals, which should be relocated in a2, might be relocated in a1. The same situation is also valid for a3, where $rand_2$ cannot be directly defined in interval $[-r, r]$. Otherwise, an individual can be relocated in any regions of the circle.

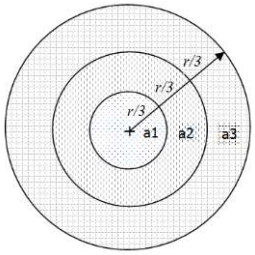


Figure 3.2 An overview of inner, middle and far circles, generated in partial relocation

```
for all individuals of a colony  **relocate all
        rand_1 ∈ [0,1];
        if rand_1 ≤ 0.11  **individual is relocated in a1
                for all dimensions
                        rand_2 ∈ [−1/3r, 1/3r]
                        new dimension:=current dimension + rand_2;
                end for
        end if
        if rand_1 > 0.11 and rand_1 ≤ 0.44  **individual is relocated in a2
                for all dimensions
                        rand_3 ∈ [0,1];
                        if rand_3 ≤ 0.5
                                rand_2 ∈ [−2/3r, −1/3r];
                        else
                                rand_2 ∈ [1/3r, 2/3r];
                        end if
                        new dimension:=current dimension + rand_2;
                end for
        end if
        if rand_1 > 0.44  **individual is relocated in a3
                for all dimensions
                        rand_3 ∈ [0,1];
                        if rand_3 ≤ 0.5
                                rand_2 ∈ [−r, −2/3r];
                        else
                                rand_2 ∈ [2/3r, r];
                        end if
                        new dimension:=current dimension + rand_2;
                end for
        end if
end for
```

Figure 3.3 A pseudo code for relocation of existing colonies

3.3.3 Making Use of Chaotic Maps

Particularly in DOPs, faster detection of higher quality regions and tracing them should be given the first priority. However, sometimes those promising regions can appear near to hard-to-detect regions like boundaries of the problem space. In the proposed approach, to overcome this difficulty, several chaotic maps along with traditional uniform random numbers are employed in order to provide an enhanced diversification capability to the explorer population at the initialization stage.

Introduced by Lorenz (1963), chaos has found numerous practical applications in many scientific fields. Mingjun & Huanwen (2004) define chaos as an irregular motion, seemingly unpredictable random behavior exhibited by a deterministic nonlinear system under deterministic conditions. Furthermore, for a chaotic system, slight differences in the initial setup might yield to great differences in the final output.

There exist several chaotic number generators reported in the literature. Among these generators, four chaotic maps, namely Chebyshev, ICMIC, tent and logistic maps are utilized in mpcFA. Moreover, for mpcFA, a modification of these chaotic maps, which generates an inverse pattern for the selected map, is also developed. In all chaotic maps, x_n represents the predecessor, while x_{n+1} denotes the succeeding number.

3.3.3.1 Chebyshev Map

Chebyshev chaotic map is generally used in neural networks, digital communication and security problems (He, He, Jiang, Zhu & Hu, 2001; He, Zhou, Xiang, Chen, & Qin, 2009). This map is formally defined by the following equation:

$$x_{n+1} = \cos(\varphi \cos^{-1} x_n) \qquad \varphi > 0, x_n \in [-1,1] \tag{3.5}$$

As it can be seen from Equation 3.5, Chebyshev chaotic map can produce chaotic numbers in [-1,1], which means that some of the individuals might be initialized out of the problem boundaries. Whether they are attracted by the inbound individuals, they gradually come closer to the boundaries, thus detecting the peaks closer to those

boundaries become easier. This is the main reason why this map is employed in mpcFA.

3.3.3.2 Logistic Map

Introduced by May (1976), logistic map is often cited as an example of how complex behavior can arise from a very simple nonlinear dynamical equation. Logistic map is a polynomial function, which generates chaotic sequences in (0,1). This map is formally defined by the following equation:

$$x_{n+1} = \varphi x_n(1 - x_n) \qquad 0 < \varphi \leq 4, x_n \in (0,1) \tag{3.6}$$

Inverse of logistic map, proposed in this thesis is obtained by subtracting x_{n+1} from unity. However, for generation of the following chaotic number, x_{n+1} is used as the input.

3.3.3.3 Tent Map

Introduced by Ott (2002), tent chaotic map is very similar to the logistic map, which displays specific chaotic effects. Tent map generates chaotic sequences in (0,1). This map is formally defined by the following equation:

$$x_{n+1} = \begin{cases} \dfrac{x_n}{0.7} & x_n < 0.7 \\ \left(\dfrac{10}{3}\right) x_n(1 - x_n) & otherwise \end{cases} \qquad x_n \in (0,1) \tag{3.7}$$

Inverse of tent map, proposed in this thesis, is simply obtained by subtracting x_{n+1} from unity again. However, for generation of the following chaotic number, x_{n+1} is used as the input.

3.3.3.4 ICMIC Map

The reason for choosing this map is similar to the reason of choosing Chebyshev map. Iterative Chaotic Map with Infinite Collapses (ICMIC) is formally defined by the following equation (He et al., 2001):

$$x_{n+1} = \sin\left(\frac{\varphi}{x_n}\right) \qquad \varphi \in (0,\infty) x_n \in (-1,1) \tag{3.8}$$

3.3.3.5 Piecewise Chaos

In this dissertation, the method used by Baykasoğlu & Ozsoydan (2015a) is adopted. Referred as piecewise chaos, where several chaotic maps are used simultaneously, this method first assigns probabilities to each of the chaotic maps. Next, piecewise intervals are generated as presented in Table 3.1. Because five different maps (including random number generator) are used here, for a fair usage, each of these maps is assigned with 0.2 probabilities. Moreover, in logistic and tent maps, 50% chance is given to perform canonical or inverse modifications. Optionally, other maps are not given the opportunity to perform inverse modifications.

In this method, when a chaotic number is required, a random number $rand \in [0,1]$ is generated first and the interval which *rand* falls into is found. Then simply the related map is used. For example, if $rand$ is 0.56, all individuals of the explorer is initialized by using the inverse logistic map. Similarly, if it is equal to 0.92, random number generator is used.

Table 3.1 Lower and upper bounds for piecewise chaos utilization

	lower bound		*upper bound*
Chebyshev map	0.00	$\leq rand <$	0.20
ICMIC map	0.20	$\leq rand <$	0.40
Logistic map	0.40	$\leq rand <$	0.50
Inverse logistic map	0.50	$\leq rand <$	0.60
Tent map	0.60	$\leq rand <$	0.70
Inverse tent map	0.70	$\leq rand <$	0.80
Random number generator	0.80	$\leq rand \leq$	1.00

The piecewise chaos of Baykasoğlu & Ozsoydan (2015a) is further improved here by making use of inverse patterns of logistic and tent maps. The aim in using this approach is to obtain a superior diversity among the individuals of the explorer population through the initialization stage. It is worth stressing that, this method is only used for the explorer population, whenever it is re-initialized due to reasons either detecting/colonizing a peak or detection of a dynamic event. In rest of the

evaluations, chaos is not utilized in any part of the algorithm. Initialization patterns of the explorer population by using each of the used chaotic maps are depicted in Figure 3.4. The rectangle in this figure represents the problem boundaries.

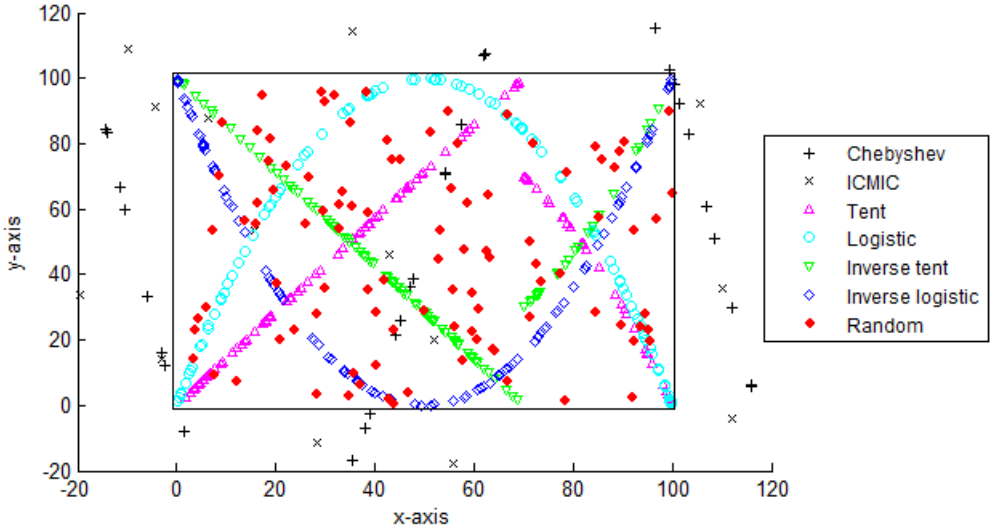


Figure 3.4 Location of the explorer by using several chaotic maps

Visualized trial tests show that the use of chaos at the initialization stage of the explorer population has benefits in finding the locations of the existing peaks. Particularly, by making use of this approach, the fireflies attracted from out of the boundaries of the problem domain, can easily detect the hard-to-detect promising regions, particularly located near to the boundaries of the problem domain.

3.3.4 Local Search

For a further improvement in the solution quality, a local search procedure is used in mpcFA. In order to keep the pressure on CPU at acceptable levels, this procedure is only applied for the best individual among all existing colonies, depending on a probability. In this method, at the end of each generation, a random number $\in [0,1]$ is generated. Local search is applied only if this number is less than *local_prob*. This procedure is followed until the completion of *local_UB* iterations. A solution is updated only if it is improved at any step throughout *local_UB* iterations.

In order to generate neighboring solutions of the best individual, the method proposed by Baykasoğlu (2012) is adopted here. Neighborhood generation of this

method is formulated in Equation 3.9, where, $coord_{jk}^{n}$, $coord_{jk}^{*}$, $rnd(\,)$ and v_k represent the *j*th coordinate (dimension) of the neighboring solution at iteration *k*, *j*th coordinate of the best individual at iteration *k*, random number generator and step size at iteration *k*, respectively.

$$coord_{jk}^{n} = coord_{jk}^{*} + [(rnd(\,) - 0.5) \times v_k] \tag{3.9}$$

Using the formulation given in Equation 3.10, step size $v_k$is systematically decremented throughout local search procedure. Thus, at the initial stages, a larger neighborhood search, whereas through the end of the search, a more precise search is performed. The parameter φ is adopted from Baykasoğlu (2012) and is fixed to 0.001 in all evaluations. The initial value of v_k is fixed to unity.

$$v_k = v_k - e^{\left(-\frac{k}{k+1}\right)} * \varphi * v_k \tag{3.10}$$

3.3.5 Lévy Flights

The final improvement to mpcFA is performed by employing Lévy flights (Shlesinger, Zaslavsky & Frisch, 1995), which implements a similar path to fruit flies or paths punctuated by a sudden 90^0 turn. Lévy flights is adopted in numerous bio-inspired algorithms as mentioned in Yang (2010).

In movements of fireflies, the last term of Equation 3.4 ($\alpha(\psi - 0.5)$) provides randomness based on continuous uniform distribution. In order to further improve the variation through this movement, Lévy flight is employed along with the traditional move of FA. Therefore, whenever a firefly is moved, either Lévy flights move (Equation 3.11) or traditional move of FA (Equation 3.4) is called with equal probabilities.

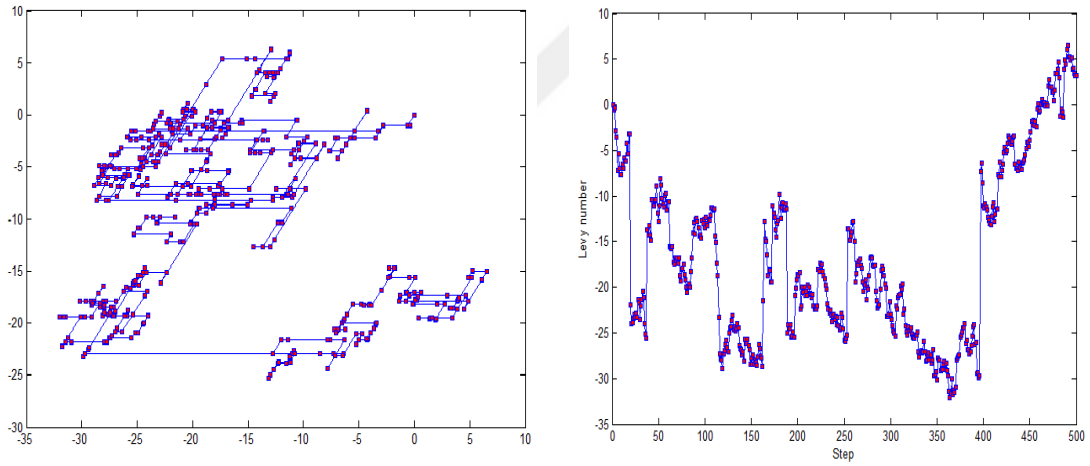
$$x_{pj} = x_{pj} + \beta_{(0)} \left({}^{1}/_{(\Omega + r)}\right) \left(x_{qj} - x_{pj}\right) + \alpha \times l\acute{e}vy \tag{3.11}$$

In this technique, the *Lévy* term in Equation 3.11 corresponds to the output (x_{n+1}) in Equation 3.12. For the next number generation, this value becomes an input (x_n),

and this recursive procedure is continued as in chaotic maps. η is fixed to 1.5 in all evaluations.

$$\omega = rnd(\,) \times 2\pi$$
$$\rho = rnd(\,)^{\left(\frac{-1}{\eta}\right)} \tag{3.12}$$
$$x_{n+1} = x_n + \rho \times \cos\omega$$

It is obvious that the formulation presented in Equation 3.12 might generate outputs, out of the interval [-1, 1]. In such cases, the related output is recurrently divided by 10, until it falls into the range [-1, 1]. However, instead of the value obtained by recursive division operations, the output x_{n+1} is directly used as the input for the next number generation. An arbitrary Lévy flight path and the corresponding values at each generation steps are depicted in Figure 3.5.



(a) (b)

Figure 3.5 a)A pattern for a Lévy flight b) Corresponding values at each generation step

3.3.6 Detection of Dynamic Events

In mpcFA, testing point method is used to detect dynamic events. In this method, a test point from problem space is chosen either randomly or by using some problem specific methods. At the beginning of each generation, a check to detect a possible change in the fitness value of this point is performed. If such a change between two successive generations is detected, it is concluded that a dynamic event has occurred

and all existing colonies are relocated as a response. In mpcFA, simply a random point from problem space is used as the testing point in order to detect dynamic changes.

3.4 Computational Study

As mentioned in Branke (1999b), "*Benchmarks should be simple, easy to describe, easy to analyze and also tunable in their parameters. On one hand they should be complex enough to allow conjectures to the real world, on the other hand they should be simple enough to allow to gain new insights into the working of the optimization algorithm.*" Although the proposed benchmark in Branke (1999b) is a synthetic problem, due to its insight characteristics and features, Moving Peaks Benchmark (MPB) of Branke (1999b) has been one of the most commonly used benchmarking problems in dynamic optimization community. The configuration of MPB, which is used in this section, is given in Table 3.2.

Table 3.2 Standard configuration of MPB scenerio-2

Parameter	Value	Additional tests
Number of peaks, *M*	10	1, 5, 20, 30, 50, 100, 200
Change frequency	5000	500, 1000, 2500
Height change	7.0	
Width change	1.0	
Peaks shape	Cone	
Basic function	No	
Shift length	1.0	
Number of dimensions, *D*	5	
Correlation coefficient, λ	0	
Peaks location range	[0-100]	
Peak height	[30-70]	
Peak width	[1-12]	
Initial value of peaks	50.0	

Although, standard setup for MPB is illustrated in the second column of Table 3.2, additional tests, including varying number of peaks with different change frequencies are conducted by using the parameter set presented in the third column of Table 3.2. In performance evaluations of mpcFA, 30 independent runs, each with different initial seeds are performed. Throughout these runs, problem environment is allowed to change 50 times.

In order to assess the efficiency of the proposed approach, a modification of *offline error* Branke & Schmeck (2003) is used as the performance measure. Presented in Equation 3.13, where $genMax$ is the maximum number of generations the algorithm is run, g_t^{best} and a_t^{best} are the fitness of the global best solution and the fitness of the best solution found by the algorithm at generation *t*, respectively, this metric is evaluated as the average of the difference between the fitness of the global best solution and the fitness of the best solution found by the algorithm. In other words, it is the average of all current errors over generations. It is worth stressing that in some of the published work *offline error* is used as the average over function evaluations, while in rest of the publications it used as the average over generations and there is not an agreed method, however because the structure of mpcFA is more adequate for *offline error* over generations, this method is utilized here.

$$offline\ error = \frac{1}{genMax} \sum_{t=1}^{genMax} (g_t^{best} - a_t^{best}) \tag{3.13}$$

The proposed algorithm is coded by using MATLAB© and all tests are conducted on a PC with hardware of 3.4 Ghz processor and 4 GB RAM.

3.4.1 Tuning the Parameters of mpcFA

Efficiency of any approximation algorithm heavily depends on the used parameters. Therefore, this section is devoted to tuning the parameters of the proposed approach. In this context, several pre-evaluations are performed. These results are given here so as to provide some insight about the effects of the employed parameters on mpcFA.

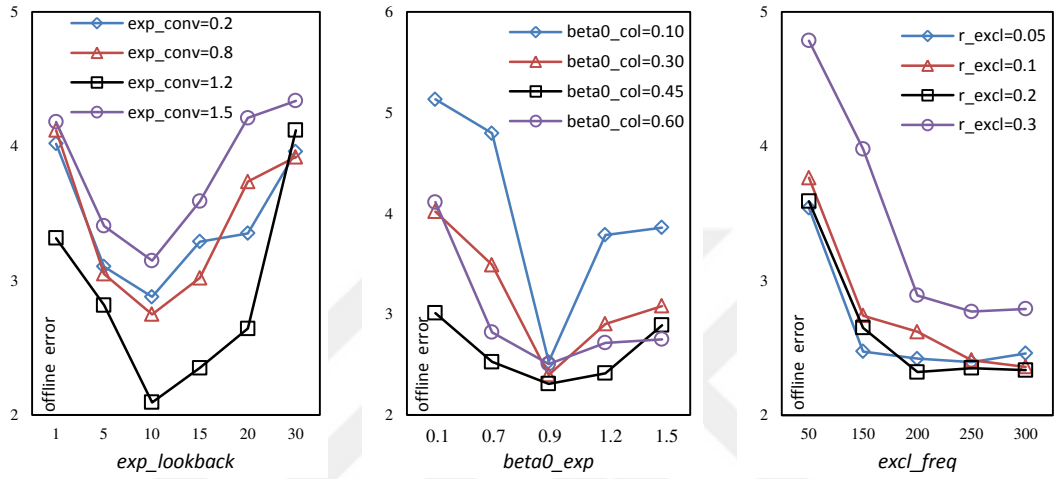
In peaks detection mechanism of mpcFA, two parameters namely, *exp_lookback* and *exp_conv* are vital to decide whether the explorer population has converged or not. Therefore, the first test is conducted in order to decide to the appropriate levels of these parameters. The analysis performed is depicted in Figure 3.6.a, where algorithm is run for each pair of *exp_lookback* and *exp_conv*. As one can see from this figure, using lower values of *exp_lookback* with higher values *of exp_conv* yields to an increment in offline error. In such cases, premature colonies are generated more

frequently, which do not contribute to the success of the algorithm. Because premature colonies are not generated closely to top of the peaks, they might never be able to converge to a peak in following generations. On the other hand, if *exp_conv* is fixed and *exp_lookback* is increased systematically, one can see that offline error decreases to a level and starts to rise again. This means that, if the explorer wastes unnecessary evaluations on an already detected peak, offline error increases because of possibly still undetected peaks. As a result, in regard to the tests performed here, the values 1.2 and 10 are found as the most promising levels for *exp_conv*, *exp_lookback*, respectively.

Another peak detection and exploitation related test is performed in order to tune the parameters $\beta_{(0)exp}$ and $\beta_{(0)col}$, which regulates the movement speed of individuals belonging to the explorer and colony populations, respectively. The effects of these parameters are reflected in Figure 3.6.b. As one can see from this figure, lower and higher values of $\beta_{(0)exp}$ yield to an increase in offline error. On one hand, if it is set lower levels, movement of the explorer population is dampened. As a result, detection of peaks takes more time. On the other hand, if $\beta_{(0)exp}$ is fixed to higher values, some of the particularly closely located peaks might be missed, which results in as an increment in offline error again. The same circumstance is valid for $\beta_{(0)col}$. An increase in offline error is observed in Figure 3.6.b for faster and slower exploitation, due to higher and lower values of $\beta_{(0)col}$, respectively. According to this test, a balance is obtained, where $\beta_{(0)exp}$ and $\beta_{(0)col}$ is fixed to 0.9 and 0.45, respectively.

In Figure 3.6.c, an analysis on exclusion parameters is conducted to obtain appropriate levels of *excl_freq* and r_{excl}. As it can be seen from this figure, frequent exclusions, where *excl_freq* is set to lower values, do not contribute to solution quality. In such cases, colonies, which have chances to diversify around the problem space are excluded along with the overlapped ones. Thus, possible higher quality solutions are also excluded while excluding non-contributing colonies. If *excl_freq* is incremented step by step, one can see that offline error approximately stabilizes over 200 generations. Moreover, higher values of r_{excl} increase the offline error again,

because in such cases, there is a probability to exclude colonies closely located but on different peaks. In other words, higher values of r_{excl} yields to erroneously exclusion of some colonies, although they do not overlap. As one can see from this figure, a good balance can be achieved, where *excl_freq* and r_{excl} is fixed to 200 and 0.2, respectively.



(a) (b) (c)

Figure 3.6 a)The effects of *exp_lookback* and *exp_conv* b) the effects of $\boldsymbol{\beta_{(0)exp}}$ and $\boldsymbol{\beta_{(0)col}}$ c) the effects of *excl_freq* and $\boldsymbol{r_{excl}}$

Presented in Table 3.3, another analysis is conducted to see the effects of parameters *col_lookback* and *col_conv*, used in standstill mode. The column *avoided* represents the number of not performed movement evaluations in mpcFA in this test. Higher values of this measure means greater gain on CPU. Accordingly, lower values of *col_lookback*, the performance of the algorithm might suffer due to freezing colonies, which have not possibly reached the highest point of their peaks yet. Thus, two criteria, namely *offline error* and *avoided*, conflict here. It should be noted that offline error has the first priority. As one can see from Table 3.3, approximate offline error values can be achieved, where *col_lookback* is fixed to 60 or 80 generations and *col_conv* is chosen as 0.05 or 0.1. Considering possible gains on CPU as a secondary criterion, according to this analysis, *col_lookback* and *col_conv* is fixed to 60 and 0.1 throughout the tests.

The effect of relocation parameter *r* is presented in Table 3.4. As one can see from this analysis, lower as well as higher values result in as increments in offline error. In regard to the obtained results, on one hand if *r* is fixed to lower values, algorithm cannot follow the trajectories of the moving optima successfully. On the other hand if it is set to greater values, colonies are relocated around a larger area, and thus they might lose the trace of an already found peak. This parameter is fixed to 3.00 throughout the tests.

Table 3.3 The effects of *col_lookback* and *col_conv*

col_lookback	*col_conv*	*offline error*	*avoided*
20	0.05	2.9337	151003
	0.1	2.9754	167765
	0.2	7.1227	171374
	0.3	7.3955	172618
40	0.05	2.2381	122697
	0.1	2.3239	136496
	0.2	3.0928	146935
	0.3	4.5351	155085
60	0.05	2.1995	104593
	0.1	2.1917	114191
	0.2	2.2006	119444
	0.3	3.0195	126369
80	0.05	2.1805	82909
	0.1	2.2059	95505
	0.2	2.3065	101860
	0.3	2.6781	107548

Table 3.4 The effect of *r*

r	*offline error*
0.3	4.3205
1.5	4.1259
2.7	2.5582
3.00	2.0513
3.60	3.0210
4.20	3.4426

The local search parameters, *local_prob* and *local_UB* are adopted from Baykasoğlu & Ozsoydan (2015a) and they are fixed to 0.35 and 20, respectively. Finally, population size of explorer and colonies are required to be determined. Population size might be increased arbitrarily aiming for better results; however,

computational speed suffers from larger population sizes, particularly on larger scaled instances. On the other hand if it chosen as too low, although computational speed increases, performance of the algorithm might deteriorate. In this respect, higher quality solutions in acceptable execution time are aimed and thus, population size of both explorer and colonies are fixed to 15 for all evaluations.

3.4.2 Performance Analysis

In this section, mpcFA is compared to the other reported approaches. It is worth stressing that in majority of the published papers, the offline error over generations is compared to the offline error over function evaluations. These two metrics indeed correspond to different comparison techniques. Therefore, in the comparison tables (Tables 3.5-3.8), only the studies with offline error over generations are given place. Offline error (standard error) of the proposed algorithm under varying number of peaks is presented in Tables 3.5-3.8.

Table 3.5 Comparison of offline error (standard error) with varying number of peaks and change frequency=500

	number of peaks							
	1	5	10	20	30	50	100	200
Hashemi & Meybodi (2009a)	9.78(0.79)	**1.91(0.18)**	**1.44(0.13)**	**2.40(0.06)**	2.90(0.08)	3.31(0.08)	3.57(0.06)	3.75(0.07)
Rezazadeh et al. (2011)	4.81(0.14)	4.95(0.11)	5.16(0.11)	5.81(0.08)	6.03(0.07)	5.95(0.06)	6.08(0.06)	6.20(0.04)
Nabizadeh et al. (2012)	8.29(0.55)	6.29(0.21)	5.45(0.17)	5.47(0.19)	5.59(0.12)	5.74(0.11)	5.45(0.07)	5.79(0.10)
Daas & Batouche (2014) (MBFO)	12.00(1.09)	8.90(0.72)	7.23(0.45)	7.38(0.38)	8.40(0.47)	7.91(0.44)	9.48(0.47)	8.39(0.35)
Daas & Batouche (2014) (a-BFO)	29.55(1.83)	17.72(0.61)	15.71(0.46)	15.68(0.42)	15.87(0.43)	15.72(0.40)	15.16(0.39)	14.81(0.35)
mpcFA	**1.23(0.05)**	2.10(0.04)	2.09(0.06)	2.45(0.07)	**2.16(0.05)**	**2.10(0.03)**	**1.94(0.03)**	**1.79(0.02)**

The results of the benchmark with the highest frequency, where dynamic events occur every 500 generations, are presented in Table 3.5. As one can see from this table, mpcFA is found as superior to other published algorithms particularly on more complex instances, where there exist more peaks. Additionally, the single peak instance under high frequency is also solved efficiently by mpcFA. For medium scaled instances with 5, 10 and 20 peaks, all of the published algorithms except Hashemi & Meybodi (2009a), are outperformed again. It is worth stressing that though the frequency of change is high here, the obtained maximum standard error is 0.07.

The results of the medium-high frequency, where dynamic events occur every 1000 generations are presented in Table 3.6. As it can be seen from this benchmark, mpcFA outperforms all of the published algorithms regardless of the number of peaks. Additionally, for this benchmark, promising standard error values are obtained again.

Table 3.6 Comparison of offline error (standard error) with varying number of peaks and change frequency=1000

	number of peaks							
	1	5	10	20	30	50	100	200
Hashemi & Meybodi (2009a)	7.37(0.53)	1.56(0.17)	1.33(0.11)	2.22(0.08)	2.74(0.08)	2.98(0.07)	3.27(0.08)	3.43(0.08)
Kamosi et al. (2010a)	4.46(0.26)	4.27(0.08)	4.61(0.07)	4.66(0.12)	4.83(0.09)	4.96(0.03)	5.14(0.08)	5.25(0.08)
Noroozi et al. (2011)	4.98(0.35)	3.96(0.04)	3.98(0.03)	4.53(0.02)	4.77(0.02)	4.87(0.02)	4.85(0.02)	4.46(0.01)
Rezazadeh et al. (2011)	2.72(0.04)	2.99(0.09)	3.87(0.08)	4.13(0.06)	4.12(0.04)	4.11(0.03)	4.26(0.04)	4.21(0.02)
Nabizadeh et al. (2012)	4.74(0.32)	3.95(0.21)	3.20(0.20)	3.52(0.17)	3.96(0.12)	3.98(0.11)	4.13(0.12)	4.15(0.01)
Daas & Batouche (2014) (MBFO)	5.88(0.40)	5.77(0.45)	6.22(0.47)	6.91(0.44)	6.57(0.42)	7.09(0.40)	7.63(0.40)	8.73(0.45)
Daas & Batouche (2014) (a-BFO)	14.48(0.90)	9.96(0.35)	10.89(0.37)	11.69(0.44)	11.50(0.43)	11.33(0.43)	11.58(0.45)	13.14(0.33)
mpcFA	**0.78(0.02)**	**0.96(0.04)**	**1.32(0.04)**	**1.72(0.07)**	**2.15(0.06)**	**1.43(0.05)**	**1.56(0.03)**	**1.39(0.03)**

The outcomes of the proposed algorithm under medium frequency, where dynamic events occur every 2500 generations, are presented in Table 3.7. As one can see from this table, the proposed approach can be considered as a promising algorithm again.

Table 3.7 Comparison of offline error (standard error) with varying number of peaks and change frequency=2500

	number of peaks							
	1	5	10	20	30	50	100	200
Hashemi & Meybodi (2009a)	6.10(0.55)	1.17(0.18)	**1.08(0.09)**	2.19(0.11)	2.62(0.11)	2.94(0.09)	3.08(0.10)	3.22(0.11)
Kamosi et al. (2010a)	1.75(0.10)	1.92(0.11)	2.39(0.16)	2.46(0.09)	2.57(0.05)	2.65(0.05)	2.72(0.04)	2.81(0.04)
Rezazadeh et al. (2011)	1.06(0.03)	1.55(0.05)	2.17(0.07)	2.51(0.05)	2.61(0.02)	2.66(0.02)	2.62(0.02)	2.64(0.01)
Nabizadeh et al. (2012)	2.31(0.21)	2.01(0.13)	1.56(0.15)	2.41(0.13)	2.78(0.10)	3.18(0.09)	3.22(0.07)	3.09(0.12)
Geshlag & Sheykhzadeh (2013)	0.87(0.01)	4.27(0.18)	4.23(0.11)	2.32(0.15)	2.55(0.04)	2.18(0.23)	2.14(0.17)	2.16(0.28)
Daas & Batouche (2014) (MBFO)	4.67(0.37)	5.76(0.49)	5.91(0.40)	6.32(0.38)	5.82(0.31)	5.90(0.33)	6.03(0.31)	5.89(0.24)
Daas & Batouche (2014) (a-BFO)	3.27(0.18)	6.10(0.40)	8.48(0.40)	8.33(0.38)	9.35(0.36)	9.76(0.36)	10.71(0.33)	11.58(0.38)
mpcFA	**0.49(0.02)**	**1.13(0.04)**	1.18(0.07)	**1.50(0.09)**	**1.20(0.05)**	**1.15(0.03)**	**1.07(0.04)**	**1.07(0.02)**

As presented in Table 3.8, a parallel finding is attained on low frequency benchmark, where dynamic events occur every 5000 generations. All reported results are outperformed in this benchmark.

In some studies, *best-error-before-change* metric (Equation 2.10) is referred as offline error. The related results are presented in Table 3.9. According to these results, the proposed algorithm achieves the best published results in larger scaled instances, which is a parallel finding to the former analyses. Additionally, in small and medium scaled instances, the majority of the reported approaches are outperformed by mpcFA again.

Table 3.8 Comparison of offline error (standard error) with varying number of peaks and change frequency=5000

	number of peaks							
	1	5	10	20	30	50	100	200
Hashemi & Meybodi (2009a)	5.23(0.47)	1.09(0.22)	1.14(0.13)	2.20(0.12)	2.67(0.13)	2.77(0.13)	2.97(0.14)	3.14(0.12)
Kamosi et al. (2010a)	0.87(0.05)	1.18(0.04)	1.42(0.04)	1.50(0.06)	1.65(0.04)	1.66(0.02)	1.68(0.03)	1.71(0.02)
Rezazadeh et al. (2011)	0.53(0.01)	1.05(0.06)	1.31(0.03)	1.69(0.05)	1.78(0.02)	1.95(0.02)	1.95(0.01)	1.90(0.01)
Nabizadeh et al. (2012)	1.02(0.14)	0.99(0.15)	1.75(0.10)	1.93(0.11)	2.28(0.10)	2.74(0.10)	2.84(0.12)	2.69(0.08)
Daas & Batouche (2014) (MBFO)	3.63(0.25)	3.85(0.41)	4.05(0.37)	4.11(0.28)	3.99(0.33)	4.97(0.27)	5.64(0.35)	6.38(0.33)
Daas & Batouche (2014) (a-BFO)	1.70(0.08)	5.67(0.37)	6.81(0.43)	8.15(0.42)	9.09(0.40)	9.59(0.35)	10.39(0.32)	10.85(0.37)
mpcFA	**0.40(0.02)**	**0.74(0.06)**	**0.90(0.05)**	**1.19(0.07)**	**1.10(0.04)**	**0.85(0.02)**	**0.84(0.02)**	**0.73(0.02)**

Table 3.9 Comparison table for *best-error-before-change* metric, with varying number of peaks and change frequency=5000

	number of peaks							
	1	5	10	20	30	50	100	200
Li & Yang (2008)	0.4e-2	-	0.18(-)	0.50(-)	0.69(-)	0.93(-)	0.90(-)	1.18(-)
Yang & Li (2010)	0.14(0.11)	0.72(0.30)	1.05(0.24)	1.59(0.22)	1.58(0.17)	1.54(0.12)	1.41(0.08)	1.24(0.06)
Li & Yang (2012) CGAR	2.02(0.05)	2.56(0.10)	2.60(0.13)	3.66(0.14)	3.12(0.10)	3.26(0.11)	2.68(0.07)	2.39(0.07)
Li & Yang (2012) CDER	0.90(0.20)	8.02(0.34)	5.52(0.16)	7.49(0.27)	5.51(0.12)	5.79(0.15)	4.12(0.10)	3.71(0.11)
Li & Yang (2012) CPSOR	0.03(0.00)	0.54(0.04)	0.59(0.04)	0.79(0.05)	1.05(0.06)	0.98(0.05)	1.06(0.04)	0.94(0.04)
Yazdani et al. (2013)	**3.48e-05 (9.01e-06)**	**0.20(0.10)**	0.25(0.05)	**0.42(0.05)**	0.48(0.04)	0.61(0.03)	0.83(0.04)	0.91(0.06)
Li et al. (2014)	0.13(-)	0.42(-)	**0.13(-)**	1(-)	**0.46(-)**	0.96(-)	1.2(-)	1(-)
Yazdani et al. (2014)	0.38(0.06)	0.55(0.04)	0.90(0.03)	1.25 (0.06)	1.47(0.05)	1.65(0.05)	1.83(0.05)	1.84(0.05)
mpcFA	0.23(0.01)	0.46(0.05)	0.67(0.05)	0.98(0.03)	0.83(0.04)	**0.60(0.02)**	**0.60(0.02)**	**0.52(0.02)**

Convergence behavior of mpcFA on an arbitrarily generated two dimensional landscape with 10 peaks is illustrated in Figure 3.7. The markers denoted by stars represent the explorer population while blue diamonds with red boundaries symbolize the fireflies of the corresponding colonies. As it can be seen from this figure, all existing peaks are detected by the proposed algorithm in less than 700 generations.

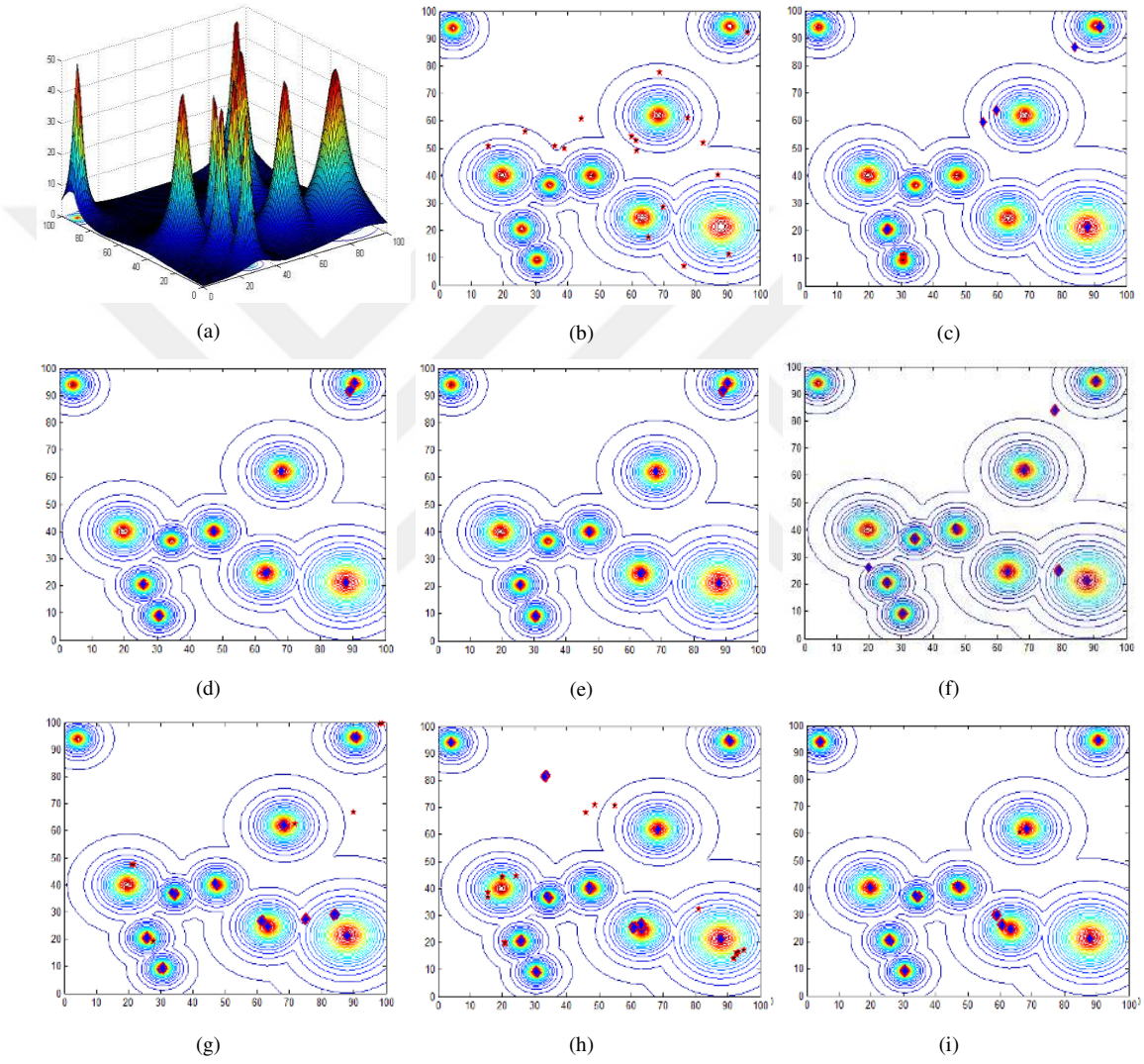


Figure 3.7 An illustration of a)an arbitrary landscape with 10 peaks, b)explorer population at the initialization stage, explorer and colonies at the end of c)100 generations, d)200 generations, e)300 generations, f)400 generations, g)500 generations, h)600 generations, i)700 generations

Additionally the convergence graphs of mpcFA for the benchmarks with 500, 1000, 2500 and 5000 frequencies are depicted in Figure 3.8. As one can seen from

this figure, the proposed algorithm has promising peaks detection and convergence capability considering the obtained superior results in the first environment. In the following environments, although mpcFA occasionally loses the track the moving optima, it is able to recover fast and find the promising regions again.

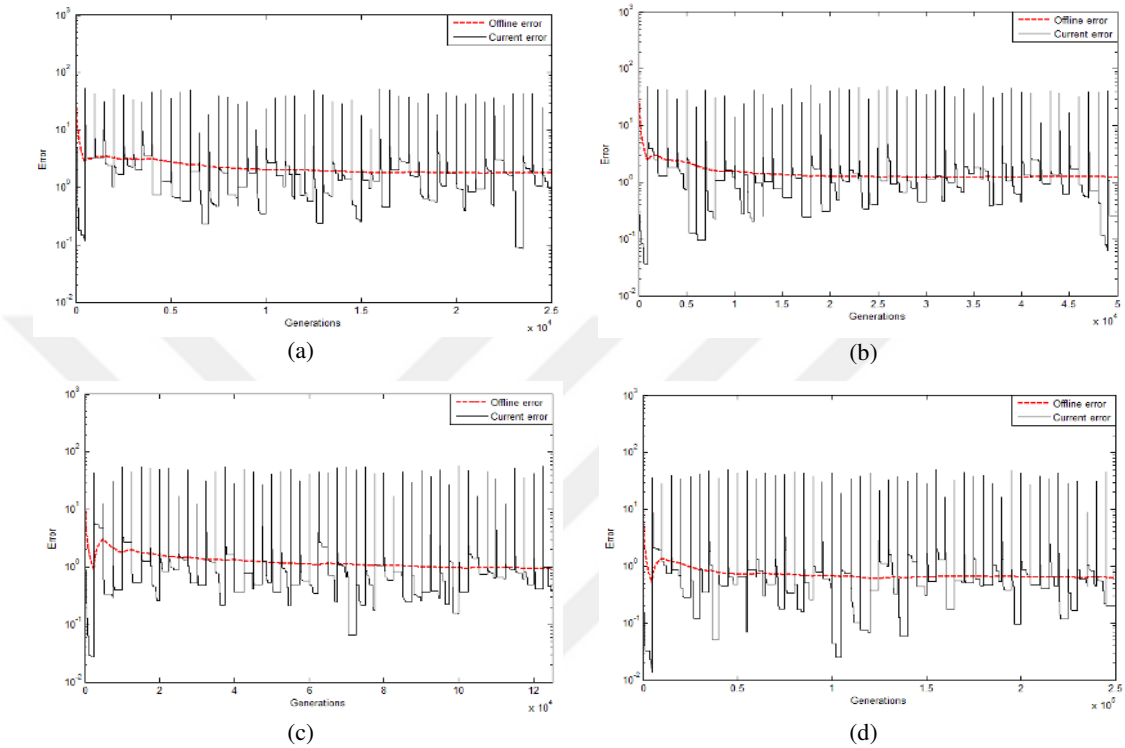


Figure 3.8 An illustration of offline error and current error of mpcFA on randomly chosen instances, where change frequencies is set to a)500, b)1000, c)2500 and d)5000

3.5 Chapter Conclusion

In this chapter, a multi-population Firefly Algorithm is proposed to solve multi-modal dynamic optimization problems, which is a hot and competitive research area.

The proposed method (mpcFA) is further improved by employing chaotic maps along with Lévy flights in order to obtain superior diversity and to increase exploration capability of mpcFA. Accordingly, an easier detection of hard-to-detect promising regions, possibly appear on the boundaries of the problem space is aimed. In order to prevent overcrowded peaks, exclusion strategy is employed in mpcFA. Additionally, a standstill mode, where non-contributing evaluations are terminated for a period, is adopted to obtain some possible gains on CPU. As another

improvement in the proposed approach, a new relocating strategy is developed. According to this strategy, subsequent to detection of dynamic events, colonies are relocated around the concentric regions, in regard to a user supplied radius. Finally, a local search procedure is employed for a better exploitation of promising regions.

Performance tests of the proposed approach are conducted on the Moving Peaks Benchmark, which is one of the most commonly used benchmarking problems in dynamic optimization community. As one can see from the experimental study, regardless of the frequency of change, mpcFA outperforms most of the reported results. Furthermore, it is seen that mpcFA performs better in more complex instances with greater number of peaks.

CHAPTER FOUR
EVOLUTIONARY AND POPULATION BASED METHODS VERSUS CONSTRUCTIVE SEARCH STRATEGIES IN DYNAMIC COMBINATORIAL OPTIMIZATION

This chapter investigates the performance of constructive and multi-start search strategies in comparison to widely used evolutionary and population based approaches on DCOPs with both dimensional and non-dimensional changes. The tests are conducted on dMKP, which has various applications in today's manufacturing and service systems.

4.1 Chapter Introduction

One of the notable efforts in dynamic optimization community is to modify the well-known EAs, which have been widely used for static optimization problems (Abdunnaser, 2006; Branke, 2001a; Jin & Branke, 2005; Liekens, 2005; Morrison, 2004; Wilke, 1999). However, as stressed before, adapting EAs to DOPs has several difficulties like inevitable convergence feature of these algorithms and possible problems particularly in solution encodings. Therefore, researchers from dynamic optimization community have proposed several approaches to handle with the inherent challenges of DCOPs.

Introduced by Taylor, Weber & Bojduj (2006), Dynamic Tabu Search (DTS), can be considered as one of the state-of-the-art methods proposed for DCOPs. DTS is capable of handling dynamic events such as arrivals, removals or modification of entities. In DTS, if a slight change occurs, the majority of the solution is re-used whereas the remaining inconsistent part of the solution is repaired. On the other hand, if a severe change is detected, the algorithm is recommended to be reinitialized with its initial configuration.

As another outstanding strategy, one can note that multi-agent based systems are extensively employed particularly in DCOPs. According to this technique, the main system is decentralized into sub-systems and each subsystem is controlled separately

by one controller. A recent classification of agent-based approaches is presented in Baykasoğlu & Durmuşoğlu (2014). Additionally, Baykasoğlu, Kaplanoğlu, Erol & Şahin (2011) and Erol, Sahin, Baykasoğlu & Kaplanoğlu (2012) report applications of agent-based systems for combinatorial problems. However, this technique requires specialized software, devoted to this system and the main purpose and/or solution strategy is not necessarily an optimization based approach; negotiation and market oriented programming procedures are mostly utilized for problem solving.

Although there is a variety of specialized methods as reviewed in the previous chapters, reported studies in the related literature abundantly employ adaptations of the well-known evolutionary based, population based or trajectory based search strategies, to make them usable in DOPs. However, this clearly requires additional effort, which has not been sufficiently debated in reported works. On the other hand, making use of constructive algorithms might have remarkable benefits in handling DCOPs.

First, in EAs, a dimensional change may cause severe problems in solution representations, as they usually utilize fixed length chromosomes. A solution representation might become unusable subsequent to a dynamic event with dimensional change. In such cases, intuition suggests discarding the cancelled part of the solution from solution representation or appending the revealed parts, which means a further repairing process. As a result, inheritance procedures (crossover, selection, reproduction, etc.) of EAs might fail to perform their tasks effectively, which also may yield to breakdowns in the mainstream of these algorithms. Crossing two chromosomes with different lengths, just before and after a dynamic event, is an example for this circumstance. Therefore, constructing a solution step by step with available parts of the solution is clearly easier than encoding whole solution in a single row.

As secondarily, an EA collects information throughout generations by using the available parts of a problem space. Assuming that, if those parts had never existed from the beginning, the algorithm would have probably converged to different regions of the search space. Therefore, discarding the cancelled parts from solution

representation might not be an efficient approach, because, gathered evolutionary information is partially lost in this strategy.

This circumstance is clearly underlined by Branke (1999a) and Branke (2001a) in the early era of the dynamic optimization as: "*If the dynamics of the optimization problem affect the genetic representation, it is not possible to simply keep old individuals, the individuals have to be adapted. For example in job shop scheduling, when new additional jobs arrive, they have to be represented in the genotype. However the adaptation of the individuals is usually rather straightforward and introduces additional variance that automatically stimulates exploration*.". Additionally, in a recent dissertation (Mavrovouniotis, 2013), the same incident is similarly stressed as in the following: "*The environmental changes are classified into two types: dimensional and non-dimensional changes, where both of them cause the optimum to change on every environmental change. Dimensional changes correspond to adding or removing variables from the problem. Such environmental changes affect the representation of the solutions and alter a feasible solution to an infeasible one. A repair operator may address this problem, but requires prior knowledge of the problem and the detection of the dynamic changes. Non-dimensional changes correspond to the change of the variables of the problem. Such environmental changes do not affect the representation of the solutions, and, thus, are more straightforward to address*.".

There exist two commonly used constructive search algorithms in the literature namely, Ant Colony Optimization (ACO) (Dorigo, 1992) and Greedy Randomized Adaptive Search Procedure (GRASP) (Feo & Resende, 1995). Although they are both constructive, the main strategy is quite different. In ACO, populations of consecutive generations are dependent to each other. This means that the final population indeed evolves from its ancestors whereas in GRASP, at each generation a new solution is re-constructed from scratch so that this strategy is additionally referred as multi-start. In other words, though they are both constructive algorithms, any additional efforts like repairing or modifying a solution to handle with dynamic events are still required in ACO, whereas in GRASP it is not necessarily required due to its multi-start strategy in its nature. Although there exists a number of ACO related

studies in dynamic optimization (Mavrovouniotis, 2013), it is still a developing field. What is more, GRASP related studies are far lacking. This is a considerable gap for the related literature, because due to its mentioned beneficial features, GRASP is one of the unique approaches, where a modification, repair or revision are not necessarily required.

One related work is addressed in Randall (2005), where a generalization of ACO to DOPs and a preliminary work on dMKP are reported. As a response to dynamic events, a strategy, based on discarding the items yielding to infeasibility and adding new ones in order to increase the profit of the knapsack is proposed in that study. Li (2011) proposes a parallel multi-start search approach for dynamic travelling salesman problem (DTSP), where a number of randomly re-initialized solutions are improved via some local search heuristics. Each of these independently generated solutions represents mostly diversified local optima, which form a promising region possibly around the global optimum. Thus, the author seeks for global optimum calculating the hit frequencies of the promising edges in each of the local optima. Although this strategy is referred as a multi-start technique, it is not related to a GRASP based approach. Mavrovouniotis (2013) presents a comprehensive survey of ACO for both static and dynamic optimization problems. In that study, an extension of ACO is employed for DTSP and dynamic vehicle routing problem (DVRP), where in DTSP non-dimensional changes and in DVRP, both dimensional and non-dimensional changes are allowed. The author employs several repair and update procedures along with immigrant schemes to handle with dynamism. Finally, Elhassania, Jaouad & Ahmed (2014) propose a genetic algorithm for DVRP, where working date is partitioned into time slices and each of these time slices represents a static VRP. Dynamic arrivals of new orders are not fulfilled during the execution of the established routes and they are postponed until following time slice. They compare their approach with GRASP and ACO. The same problem with similar approaches is reported in Montemanni et al. (2003) and Montemanni, Gambardella, Rizzoli & Donati (2005). Some real-life examples, adopting GRASP in online decision making are presented in Argüello, Bard & Yu (1997), Khadidja, Fatima & Fayçal (2012) and Campbell & Savelsbergh (2005).

Most of the studies published so far are focusing on making an algorithm to be capable of keeping track of promising regions and not to forget previously found high quality solutions or parts of those solutions. For example, the aim in making use of additional repair or revising strategies for EAs in DOPs is indeed to preserve the gathered evolutionary information and to use them in the new environment. However, there rise some questions like "what if the strategy of an algorithm is based on forgetting the previously found solutions and re-generating new solutions with only currently available data in a constructive way".

The main motivation of this chapter is to provide insight about the use of multi-start and constructive search strategies in DCOPs to overcome the aforementioned difficulties of this domain and thus to present an alternative approach, which is easily applicable to other various DCOPs with both dimensional and non-dimensional changes. Therefore, in order to demonstrate the performance of such strategy, a comparison analysis of GRASP with some state-of-the-art methods including GA, DE and a relatively more recent optimizer FA, is conducted on dMKP, which has been widely used as a benchmarking environment in DOPs. Moreover, a hybridization method of GRASP with a population based algorithm is also provided in this chapter.

In this context, the rest of this chapter is organized as follows: static and dynamic versions of the MKP are presented in Section 4.2, the proposed solution approaches and performance analyses are given in Section 4.3 and Section 4.4, respectively. Finally, chapter concluding remarks are provided in Section 4.6.

4.2 Multi-dimensional Knapsack Problem and its Extension with Dynamism

In this chapter, dMKP is selected as the benchmarking environment because it is closely related to many real-life optimization problems such as cargo loading, selecting project funds, budget management, cutting stock, delivery of groceries in vehicles with multiple compartments, capital budgeting, allocating processors and data in distributed computer systems, etc. Therefore, a developed and tested algorithm for dMKP has a chance to be generalized to other various DCOPs.

The famous static case of MKP is known to be NP-complete and according to Kellerer, Pferschy & Pisinger (2004), knapsack problems have been widely used as combinatorial benchmark problems of EAs (Branke et al., 2006; Branke, Salihoğlu & Uyar, 2005; Rohlfshagen & Yao, 2009; Uyar & Uyar, 2009). A common mathematical formulation of the static MKP is given as follows in the literature:

$$maximize \sum_{j=1}^{n} p_j x_j \tag{4.1}$$

$$s.t. \qquad \sum_{j=1}^{n} w_{ij} x_j \leq c_i \qquad \forall i \in I \tag{4.2}$$

$$x_j \in \{0,1\} \qquad \forall j \in J \tag{4.3}$$

where $J = \{1,2,3,\dots,n\}$ is the set of items, $I = \{1,2,3,\dots,m\}$ is the set of dimensions of knapsack, n is the number of items, m is the number of dimensions of the knapsack, x_j is the binary variable denoting whether the *j*th item is selected or not, p_j is profit of the *j*th item, c_i is the capacity of the *i*th dimension of the knapsack and w_{ij} is the unit resource consumption of the *j*th item for the *i*th dimension of the knapsack and all parameters are assumed to be positive. Because MKP and its base version single dimensional 0/1 knapsack problem have an extensive literature, only the related work considering dynamic extensions of these problems is given here.

Goldberg & Smith (1987) introduce a problem referred as time-varying knapsack problem, where the permissibly capacity of the knapsack is allowed to change over time (Goldberg, 1989; Mori, Kita & Nishikawa, 1996; Ryan, 1997). In recent years, He et al. (2016) introduce a new problem, referred as randomized time-varying knapsack problem, which is further extended from time-varying knapsack problem of Goldberg & Smith (1987), where in this recently introduced problem, additionally profits and weights of the items as well as the knapsack capacity change periodically. Feng & Wang (2015) also handle the same problem, where several population-based algorithms are used as solution approaches. Smith & Vavak (1999) report a work, where number of items as well as associated profits remains stationary and weights alter after a predefined period. They employ GA with several extensions as solution

approach. A stochastic/dynamic knapsack problem, where items arrive dynamically w.r.t. a Poisson distribution is reported in Kleywegt & Papastavrou (1998) and Kleywegt & Papastavrou (2001). In this problem, the aim is to maximize the profit of accepting appropriate items, while rejecting rest of them, where rejection causes penalty. The same problem is used in Hartman & Perry (2006). Van Hemert, van Hoyweghen, Lukschandl & Verbeeck (2001) employ a GA, extended with an inference strategy, which tries to predict the upcoming environments for dKP. In the same year Simões & Costa (2001) propose a specialized crossover operator for the same problem. The same authors (Simões & Costa, 2002) test the effects of transformation in EAs in comparison with hyper-mutation and random immigrants on the same problem. Later, Simões & Costa (2007a; 2007b) address the same problem using an EA, based-on memory and maintaining diversity schemes, frequently employed in DOPs community. The same authors (Simões & Costa, 2009c) further extend this study by investigating the effects of population and memory sizes. In the same year Simões & Costa (2009a; 2009b) propose prediction techniques, based on regression models to solve 0/1 dKP. Yang & Yao (2003) employ dualism integrated population based incremental learning algorithms for the same problem. Karaman & Uyar (2004) introduce a dynamic event detection method and an environment quality measurement, which is evaluated as the ratio of all possible feasible solutions to all possible solutions of the current problem space. Basically, change in the magnititude of this ratio gives some clues about the severity of the dynamic event. Rand & Riolo (2005) test the effectiveness of GAs on different benchmarking problems including dKP. An alternative splicing method to be used in GA is proposed by Rohlfshagen & Bullinaria (2006). Their method is tested on a 0/1 dKP, where only the profits and weights of some randomly selected items are changed and capacity of the knapsack remain stationary throughout evaluation process. In the same year, Klinkmeijer, de Jong & Wiering (2006) introduce an extension of GA, enhanced with memory. It is further compared to some state-of-the-art methods including hyper-mutation and a diploid GA. The same problem is solved by Yang (2007), using a GA, enhanced with elitism-based immigrants, later hybridized with random immigrants method. Yang & Tinós (2008) employ another extension of GA, where selection pressure is tuned subsequent to dynamic events.

An interesting work, combining an agent-based and evolutionary strategy, which is further improved by integrating random immigrants for 0/1 dKP is proposed by Yan, Wang, Wang & Wang (2008). Alba, Luque & Arias (2009) focus on the influence of severity and the frequency factors on the same problem. Wang, Yang, Ip & Wang (2009) propose a primal-dual genetic algorithm to solve dKP. Yang & Richter (2009) investigate the effects of the learning rate for population-based incremental learning on the same problem. A multi-population based approach with a probabilistic strategy is proposed by Wu, Wang & Liu (2010). Rohlfshagen & Yao (2011) present an extended the work of Rohlfshagen & Yao (2008) in another related problem namely, the dynamic subset sum problem. The authors propose a binary encoding technique and a penalizing procedure to avoid infeasibility.

In comparison to dKP related studies, reported work, adopting dMKP as the benchmarking environment, is lacking. Branke et al. (2006) analyse the effects of solution representation techniques on dMKP. Baykasoğlu & Ozsoydan (2014) use the same problem with different assumptions, where an improved FA is employed. In one recent study, Ünal & Kayakutlu (2016), report an extension of a GA, where it is tested on both static and dynamic versions of MKP as in Baykasoğlu & Ozsoydan (2014). Uyar (2007) investigates the replacement strategies in steady-stage GAs on dMKP. Later, Perry & Hartman (2009) propose using a stochastic dynamic programming approach to solve dynamic and stochastic MKP, where items arrive in regard to a stochastic process. Uyar & Uyar (2009) stress some shortcomings of previously reported dynamic generators used in DOPs community and propose a new generator, which is taking into account the direction of the change, i.e. whether it is a relaxing or more restricting change.

Literature also includes some applications of this problem in energy saving projects. Ceran (2014) formulates the dynamic utility-maximizing allocation problem with discrete set of user requests as a dynamic and stochastic knapsack problem with randomized incremental capacity. A threshold-based policy, employing a dynamic programming strategy is used in that work. Erkılıç (2014) reports a similar problem, where additionally GAs and online heuristics are used. As another real-life application, Aisopos, Tserpes & Varvarigou (2013) model resource allocation

problem in software service systems as a fractional knapsack problem, where jobs (requests) dynamically arrive to system. Granmo, Oommen, Myrer & Olsen (2006) address a similar problem with additional non-linear objective, where a learning automata scheme is used. Kumaraguruparan, Sivaramakrishnan & Sapatnekar (2012) model the residential scheduling of power grid with dynamic pricing strategies as a dMKP.

4.2.1 Sources of Dynamism

As mentioned previously, there are two different sources of dynamism in dMKP: *time-varying parameters* (non-dimensional changes) and *domain based changes* (dimensional). An abstraction of these dynamic events is illustrated in Figure 4.1.

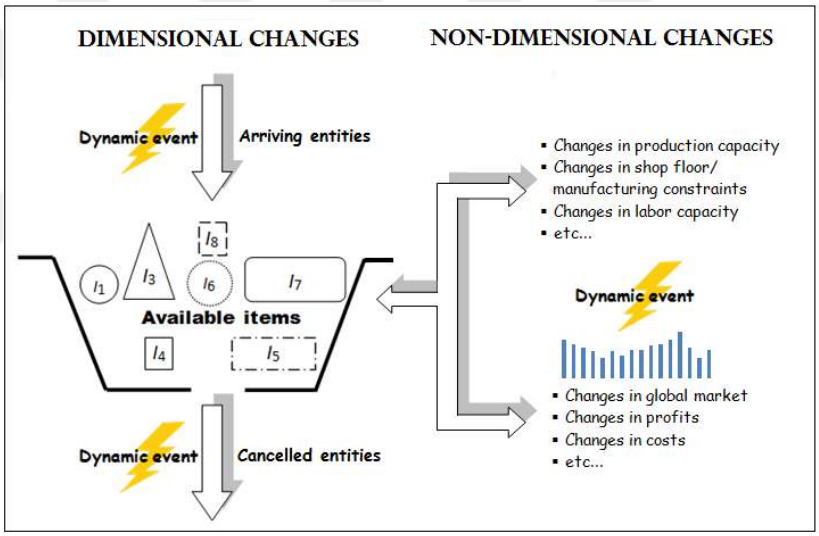


Figure 4.1 A snapshot of the dynamic environment in dMKP

Time-varying parameters generally represent the changes, related to problem parameters. This type of change (Figure 4.1) can arise due to either outer (global market) reasons like changing real-world conjuncture, prices, utilization coefficients, etc. or due to inner (production facility) effects like labor capacity, production capacity, breakdowns, etc. On the other hand, dimensional dynamic events yield to changes in problem domain like dimension changes in problems definition sets, where variables, parameters and constraints of the optimization problem are defined. In this thesis, only arrivals or cancellations of items are allowed to occur as dimensional changes. Additionally, non-dimensional changes are allowed to affect

all coefficients and right-hand-side values of the constrains. Both dynamic events might result in changes of feasible search space. Thus, a feasible solution might become an infeasible one in forthcoming environments or vice versa.

4.2.2 Designing Dynamic Benchmark Environment

Non-dimensional changes, which is the first source of dynamism, is adopted here as proposed in Baykasoğlu & Ozsoydan (2014). The problem related parameters change after a series of evaluations, which gives the *frequency* of change.

Initially a starting environment is required as a basis in order to dynamically generate consecutive environments. Therefore, the first instance of*mknapcb4.txt*[1] is used as the basis environment (Branke et al., 2006; Baykasoğlu & Ozsoydan, 2014). This instance includes 100 items and 10 dimensions. The parameters change subsequent to dynamic events, which are triggered periodically.

$$p_j = p_j * \left(1 + N\left(0, \sigma_p\right)\right) \tag{4.4}$$

$$w_{ij} = w_{ij} * \left(1 + N(0, \sigma_w)\right) \tag{4.5}$$

$$c_i = c_i * \left(1 + N(0, \sigma_c)\right) \tag{4.6}$$

The parameters of the new environment are changed via Equations 4.4-4.6, based-on the parameters of current environment. The standard deviation for each parameter is assumed to be equal as $\sigma_p = \sigma_w = \sigma_c = 0.05$.

In dimensional changes, a number of items, where this number is randomly determined between [1, 10], is allowed to arrive. The same procedure is followed to define the number of items to be removed. As a result, at the end of each dimensional dynamic event, a number of items are appended while some of the currently available items are discarded.

---

[1]http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknapcb4.txt

Basically, items to be cancelled are randomly selected. However, one critical issue is determining the profit and resource consumption parameters of the arriving items. The data of new items are not known beforehand and they are revealed to the algorithm only when they arrive. In order to define the parameters of arriving items, statistical distributions of the related parameters in *mknapcb4.txt* are used. For this purpose, two distinct fitness tests are performed using MINITAB©. The first test is performed for estimating the profits of arriving items, where the profit values of 100 items are used. In regard to minimum Anderson-Darling values and appropriate level of *p*-value, gamma distribution with 16.02969 shape and 46.83994 scale parameters achieved the best goodness of fit test values with 95% confidence level as shown in Figure 4.2. Therefore, the profits of the arriving items are assumed to be distributed as gamma distribution p_j~G(16.03, 46.84). The second test with 1000 data entries (10×100, where 10 is number of dimension of the knapsack, 100 is number of items) is performed for determining the resource consumption coefficients of arriving items. The results of the goodness of fit test demonstrated that resource consumption parameters are normally distributed as w_{ij}~N(501.803, 290.184^2). However, as it can clearly be seen, some of the random variables derived from the distribution may result in negative values. Therefore, absolute values of those variables are used throughout the study.

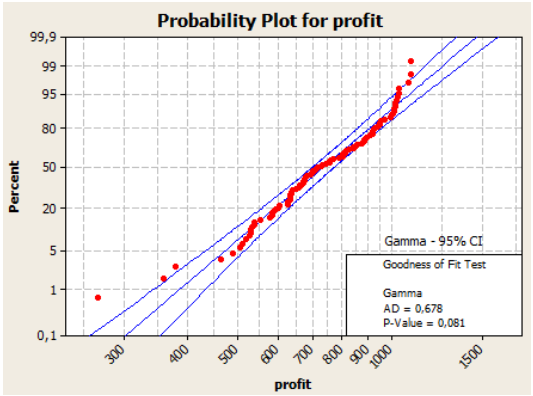


Figure 4.2 Probability plot for profit of the known items

Particularly in real-life applications, dynamic events can be triggered at any time in horizon. However, for a better visualization of the experimental study when

comparing multiple algorithms on the same plot, each dynamic event (non-dimensional and dimensional) is assumed to occur simultaneously.

4.3 The Proposed Solution Approaches

4.3.1 Constructive Strategy in Dynamic Environments

GRASP (Feo & Resende, 1995) is a multi-start strategy that uses both construction and improvement heuristics to generate high quality solutions. Differently from evolutionary approximation techniques, each generation is reinitialized from scratch. In GRASP, a solution is constructed step-by-step via a construction heuristic, determining the next item to be added to the solution by occasionally choosing a feasible item in regard to a problem specific greedy rule.

A greedy rule is a function used for sorting items w.r.t. their greedy values. The next item to be added to solution is randomly selected from among the items sorted using this rule. A parameter $\eta \in [0,1]$ imposes a restriction to selectable candidates (items). As a result, a *restricted candidate list* (RCL) is obtained. Any item, to be added to the solution, can only be selected from RCL. A larger RCL increases randomness of search, whereas a smaller size RCL gives priority to a greedy behavior in construction phase.

4.3.1.1 Constructing a Solution

As reported in Moraga (2002) and Moraga, DePuy & Whitehouse (2005), there are several greedy rules proposed for the static case of MKP. Here, the *dynamic greedy rul*e is used due to being sensitive to dynamic changes, rather than a direct solution construction in a single pass i.e. RCL is regenerated subsequent to each item selection. In this rule, penalty factor h_j for the *j*th item is computed as in Equation 4.7.

$$h_j = \sum_{i=1}^{m} \left(\frac{w_{ij}}{c_i - CW_i} \right) \qquad \forall j \in J \tag{4.7}$$

$$\acute{\rho}_j = p_j / h_j \tag{4.8}$$

where CW_i is the amount of *i*th resource consumed by the assigned items. After obtaining penalty factors of each assignable candidate items, pseudo-utility ratios $\acute{\rho}_j$ are evaluated dividing the current profits of each item to their penalty factors as given in Equation 4.8. Next, the items are sorted by non-increasing magnitude of their pseudo-utility ratios and RCL is generated involving only the items with pseudo-utility ratios in the interval $\left[\acute{\rho}^{max},\ \acute{\rho}^{max} - \eta \times \left(\acute{\rho}^{max} - \acute{\rho}^{min}\right)\right]$.

The assigned (included in the knapsack) and not assigned items are recorded in the lists shown by S and S^c respectively. Thus, at the beginning of construction phase, while S is an empty list, S^c is comprised of sorted index of items shown by $S^c = \langle 1,2,3,\dots,n\rangle$, where n is the number of available items. While constructing a solution, whenever an item is assigned to the knapsack, the index of that item is appended to the end of S and it is removed from the list S^c. To sum up, S and S^c can be considered as two tuples, listing the index of the assigned and not assigned items.

4.3.1.2 Improvement Phase

In the proposed approach, a local improvement procedure is performed after constructing a solution. The goal of this phase is to achieve possible improvements on the constructed solutions.

Local improvement phase implemented here is based on pairwise exchanges of the items listed in S and S^c. However, on one hand, while some of those pairwise exchanges might violate the capacity constraints, on the other hand, although an arbitrary exchange is feasible, it might not contribute to the total profit of the knapsack. Therefore, in improvement phase, first, all feasible and contributing pairwise exchanges from among all items, listed in tuples in S and S^c are filtered. These filtered assigned and not assigned items, whose exchange is feasible and profitable, are recorded in lists (tuples) *AI* (the list of assigned items) and *UI* (the list of unassigned items), respectively, while the possible gains are appended in sequence to the gain list, denoted by *G*. Thus, the items, whose pairwise exchanges are both feasible and profitable, are listed in *AI* and *UI*, which become actually the subsets of S and S^c, respectively. For example, given that $AI = \langle 15,15,17,9\rangle$, $UI = \langle 1,6,14,29\rangle$

and $G = \langle 10.80, 5.65, 15.10, 7.00 \rangle$, one can see that the maximum gain, which can be found by argmax(G) is the third pairwise exchange, which is obtained by swapping the third elements of *AI* and *UI*. In other words, exchange of items 17 and 14, increases the total profit of the knapsack by 15.10. Or arbitrarily, any other exchange e.g. the swap of items 15 and 1, which contribute to the total profit by 10.80 or swap of items 15 and 6, which contribute to the total profit by 5.65 can also be performed.

A detailed illustration of the proposed GRASP algorithm including both construction and improvement phases is given in pseudo codes, depicted in Figure 4.3.

In this way, it is possible to perform a first fit (exchange of AI_1 and UI_1), best fit (exchange of $AI_{argmax(G)}$ and $UI_{argmax(G)}$) or random fit swaps, on demand. However, in the proposed approach, in order to introduce diversity to the found solutions, a hybrid of random and best fit is used as local search. According to this implementation, equal chances are given to the best fit and random swaps.

4.3.2 Genetic Algorithm

GA simulates the behavior of evolution of organisms. It is one of the sate-of-the art methods, widely used as a stochastic search technique, which can find solutions to various NP-hard problems (Holland, 1975; Goldberg, 1989). GA uses evolutionary operators like crossover, mutation and selection. GA can also be considered as a basis for other EAs.

In the GA applied here, population is initialized with randomly generated chromosomes. Single point crossover is used as the crossover operator and one offspring is generated each time. Offspring has a 0.5 chance to take the first segment from the first parent and second segment from the second parent; and 0.5 chance to take the first segment from the second parent and the second segment from the first parent. Mutation operator simply assigns a new random number $\in$ [0, 1] to the bits in solution representation, which is clarified in the following. Finally, roulette wheel

selection is used in the reproduction stage of GA. A pseudo code for the mentioned GA is presented in Figure 4.4.

```
// Load test problem, configure input parameters
while gen≤genMAX
        remainig_cap_i := c_i   ∀i
        // construction phase
        while w_ij ≤ remainig_cap_i     ∀i, ∃j
                evaluate ṕ_j of each item in S^c
                RCL^lower := ṕ^max − η × (ṕ^max − ṕ^min);    RCL^upper := ṕ^max
                Select a random element π from S^c
                whose ṕ values are in [RCL^lower, RCL^upper]
                if w_iS^c_π ≤ remainig_cap_i   ∀i
                        append S^c_π to the end of S
                        remainig_cap_i := remainig_cap_i − w_iS^c_π   ∀i
                end if
                Remove the π th element of S^c
        end while
        //solution improvement phase
        is_gain :=1;
        while is_gain =1
                is_gain :=0;
                for t=1:|S|
                        for k=1: |S^c|
                                feasible :=true;
                                if remainig_cap_i + w_iS_t − w_iS^c_k < 0     ∃i
                                        feasible :=false;
                                end if
                                if feasible==true and p_k > p_t
                                        is_gain :=1;
                                        append S_t to the end of list AI
                                        append S^c_k to the end of list UI
                                        gain :=p_S^c_k − p_S_t;
                                        append gain to the end of list G
                                end if
                        end for
                end for
                if is_gain ==1;
                        r=rand ϵ [0,1];
                        if r<=0.50
                                k :=argmax(G);
                        else
                                k :=int(|G|×rand+1);
                        end if
                        remove item AI_k from S
                        append item UI_k to the end of list S
                        remove item UI_k from S^c
                        append item AI_k to the end of list S^c
                        remainig_cap_i := remainig_cap_i + w_iAI_k − w_iUI_k   ∀i
                        Update the total profit of the knapsack
                end if
        end while
        gen :=gen+1;
end while
//S:list of assigned items; S^c:list of not assigned items
//c_i:capacity of i th dimension of the knapsack
//p_k is the profit of the k th item
//G:list of gains, obtained by swaps of the elements listed in AI and UI
//AI: the set of assigned items; AI: the set of unassigned items
//remainig_cap_i:remaining capacity of i th dimension of the knapsack
//genMAX: maximum number of generations; gen: generation index
```

Figure 4.3 A pseudo code for GRASP

```
Generate initial population
Evaluate the fitness of all chromosomes in whole population
while (g<MaxGen)
        Perform crossover and mutation operators
        Evaluate the fitness of whole population
        Find the current best, update the global best if required
        Perform selection & copy
end while
Results and visualization
```

Figure 4.4 A pseudo code for GA

4.3.3 Differential Evolution Algorithm

DE (Storn & Price, 1995) is a population based metaheuristic which implements distance based evaluations between individuals in the population. As in other EAs, DE also employs inheritance operators such as crossover, mutation and selection. There are various extensions of DE (Das & Suganthan, 2011), however *DE/rand/1/bin* is discussed here.

4.3.3.1 Initialization

Similarly to other algorithms used here, DE also starts with a random population, comprised of vectors ${X_p}^{gen}$. Initial population is generated as formulated in Equation 4.9. For DE, the individual ${x_{pj}}^{gen}$ is the value of the *j*th dimension of the *p*th individual at iteration *gen*.

$$ {x_{pj}}^{1} = rand_j[0,1] \times \left(ub_j - lb_j\right) \tag{4.9} $$

where ub_j and lb_j are the upper and lower bounds of the corresponding dimensions if they exist.

4.3.3.2 Mutation

For each target vector ${X_p}^{gen}$, a mutant vector ${V_p}^{gen+1}$ is generated using the Equation 4.10.

$$ {V_p}^{gen+1} = {X_{r_3}}^{gen} + F \times \left({X_{r_1}}^{gen} - {X_{r_2}}^{gen}\right) \;\; r_1, r_2, r_3 \epsilon [1, pop\_size]; \;\; r_1 \neq r_2 \tag{4.10} $$

where r_1, r_2, r_3 are random integer numbers, representing the index of the randomly chosen individuals.

4.3.3.3 Crossover

As formulated in Equation 4.11, the mutant vector is crossed with the target vector in order to produce trial vector ${U_p}^{gen+1}$

$$ {u_{pj}}^{gen+1} = \begin{cases} {v_{pj}}^{gen+1} & if\ (rand_j \le CR) \\ {x_{pj}}^{gen} & otherwise \end{cases} \tag{4.11} $$

where $rand_j \in [0,1]$ is a random number and $CR \in [0,1]$ is the crossover rate.

4.3.3.4 Selection

Finally, fitness of the trial vector ${U_p}^{gen+1}$ is evaluated and if the fitness of ${U_p}^{gen+1}$ is better than the target vector ${X_p}^{gen}$, ${X_p}^{gen+1}$ is updated as ${U_p}^{gen+1}$, otherwise ${X_p}^{gen+1}$ is assigned as ${X_p}^{gen}$. Unlike the selection operator as in GAs, selection here gives equal chances to all trial vectors generated from target vectors. A pseudo code for DE used here is illustrated in Figure 4.5.

```
Generate initial population
Let f(X_i^g) denote the objective value of X_i^g
while (g<genMax)
    for i=1:pop_size
        Create the mutant vector V_i^{g+1}
        Apply crossover using V_i^{g+1} and X_i^g to produce U_i^{g+1}
        if f(U_i^{g+1}) is better than f(X_i^g)
            X_i^{g+1} =: U_i^{g+1}
            f(X_i^{g+1}) =: f(U_i^{g+1})
        else
            X_i^{g+1} =: X_i^g
            f(X_i^{g+1}) =: f(X_i^g)
        end if
    end for
    Find the current best, update the global best if required
end while
Results and visualization
```

Figure 4.5 A pseudo code for DE

4.3.4 Solution Encoding

The priority-based encoding technique is employed in the algorithms GA, DE and FA. In this method, the length of an encoded solution is equal to the number of items in the problem. The bits of the individuals simply hold unbounded continuous numbers (random keys), representing the priority of each item. An example for an encoded solution is presented in Figure 4.6.

0.411	0.729	0.006	0.284	0.907	0.569	0.361	0.832	0.693	0.418
1	2	3	4	5	6	7	8	9	10

Figure 4.6 An example for solution representation

In Figure 4.6, let's assume that, indexed in a range from 1 to 10 (*2*nd row) and associated with random priorities (*1*st row), 10 items exist. A higher value in *1*st row provides a higher priority for the corresponding item, unless it violates capacity constraints. According to the priorities, items 5, 8, 2, 9, 6, 10, 1, 7, 4, 3 are attempted to be assigned in sequence. For instance, let item 5 be assigned. Next, the capacities of the knapsack dimensions are decreased as much as the resource requirements of item 5. Let' s suppose that remaining capacities are not enough for item 8. This yields item 8 to be ignored and the same procedure is carried out for the rest of the sequenced items. Decoding procedure is illustrated formally in Figure 4.7.

```
//Let x_pj denote the jth dimension of the pth firefly of a population
for p=1: pop_size
        S: = ∅ ;
        remainig_cap_i: = c_i;    ∀i
        for k=1:|J|
                ω :=argmax_{j∈J}(x_pj)  where x_pj ≠ −M ;
                if w_iω ≤ remainig_cap_i   ∀i
                        S := S ∪ {ω};
                        remainig_cap_i := remainig_cap_i − w_iω ;   ∀i
                end if
                x_pω: = −M;
        end for
end for
** M is a sufficiently large number
** S is the set of items included in knapsack
** c_i:capacity of i th dimension of the knapsack
** J is the set of items, considered in the problem
** remainig_cap_i:remaining capacity of i th dimension of the knapsack
```

Figure 4.7 A pseudo code for decoding a population

As one can see from Figures 4.6-4.7, number of dimensions of each solution vector is equal to the number of items exit in problem domain. Thus, each dimension of a solution vector, in other words each item is represented by a real number (priority). The advantage of this technique is not only enabling an easier implementation of inheritance feature for EAs, but it also generates feasible solutions. Thus, any further repairing procedure to maintain feasibility is not required.

4.3.5 Hybridizing GRASP with FA

One of the existing challenges here is the structure of solution encodings of GRASP and FA. It is obvious that a solution obtained via GRASP is only a list of items included in the knapsack i.e. it does not hold information about priority values (*1*st row of Figure 4.6) as in encoded solutions of GA, DE of FA. As a result, solutions obtained via GRASP are not usable in FA with their current forms. Therefore, using the information of assigned items and their attributes, the procedure given in Figure 4.8 is utilized to transform this information to priorities to be usable in FA.

According to this procedure, priority values of the items included in the knapsack are evaluated via a simple greedy rule. However, for the rest of the items, those priority values are generated in an interval, restricted by lower and upper bounds. Next, in order not to pass identical priority vectors (encoded solutions) to FA, obtained priority vector is multiplied by a random number $r \in [1,2]$.

The hybridized version of FA and GRASP has three main stages, namely GRASP stage (diversification), transformation stage and FA stage (intensification). In this approach, at the initialization or when a change is detected, GRASP stage is triggered and a standalone GRASP searches for a period (inner iterations), defined by a parameter called *solo_search*. This stage can be considered as a procedure, which is performed only at the beginning of each changing environment and whose aim is to initialize FA with more enhanced solutions. After completing *solo_search*, transformation and selection (the second stage) are performed following the procedures depicted in Figures 4.8-4.9. At the final stage, only FA is run throughout

the rest of the generations. GRASP and transformation stages are never called again until another change is detected. If a dynamic event occurs, the same procedure GRASP-transform-FA is followed again. Both hybridized (HybGrFA) and base versions of GRASP are allowed to conduct same number of outer generations in each of the changing environments. A flow chart, representing the flow of HybGrFA is depicted in Figure 4.10. The details of the stages of HybGrFA are given next.

```
//Let Ȳ = (y_1, y_2, ..., y_t) be an arbitrary solution vector obtained via GRASP
//Each element of Ȳ is the index of items included in knapsack
priority_{1×n} := 0̄;
for k=1:t
        total=0;
        for i=1:m
                total:=total+w_{iy_k}/c_i;
        end for
        priority(y_k) :=p_{y_k}/(1000 × total);
end for
UB:= min{priority}-0.5; LB:=0;
if UB<0
        UB:= min{priority}-0.1;
        if UB<-max{ priority}      //relatively closer priority values;
                LB:=-max{priority};
        else
                LB:= min{priority}-max{priority};
        end if
end if
for j=1:n
        if priority(j) ==0
                priority(j):=(UB-LB)×rand+LB;
        end if
end for
r:=(2×rand+1); priority:= priority×r;

//c_i capacity of ith dimension of the knapsack
//w_{iy_k} is the unit resource consumption of item y_k for the i th knapsack dimension
//UB and LB are upper/lower bounds for priorities of unassigned items, respectively
//n: number of items; m: number of dimension of knapsack; p_{y_k}: profit of the item y_k
```

Figure 4.8 Generating priority values for FA

Throughout the search of the first stage, solutions found at each evaluation of GRASP are recorded in a list referred as *intelligence list*. Identical solutions are not given place and only one copy of them is allowed in this list. Next, solutions in *intelligence list* are sorted by non-increasing order and sorted solutions are ranked, where the *1*st rank represents the best solution.

```
//Obtain the intelligence list via GRASP
Sort solutions in intelligence list by non-increasing obj.
for t=1:solo_search
        log_array(t)=1+log10(t);
end for
max*:=max{log_array};
for t=1: solo_search
        log_array(t):=max*- log_array(t)+0.00001;
end for
max*:=max{log_array};
for t=1: solo_search
        log_array(t):= log_array(t) / max*;
end for
for k=1: θ × solo_search
        for t=2: solo_search
                x=rand ϵ[0,1] ;
                if x≥ log_array(t) and x< log_array(t-1)
                        Select solution t to pass to FA;
                end if
        end for
end for
//θ ×solo_search solutions are passed to FA. (θ ∈ [0,1])
//The length of intelligence list is equal to the parameter solo_search.
```

Figure 4.9 The pseudo code for passing solutions to FA

Subsequent to search of GRASP, at the second stage a random part of the information kept in *intelligence list*, is passed to FA for intensification around higher quality solutions from diverse regions of the search space. For this purpose, using the procedure, already given in Figure 4.8, priority based encodings of the corresponding solutions kept in intelligence list are generated. In what follows, those solutions to be passed to FA ($\theta \times solo\_search$) are selected using the pseudo codes, depicted in Figure 4.9. The rest of the FA population is re-initialized randomly.

As one can see from Figure 4.11, the procedure depicted in Figure 4.9 gives more chance to qualified solutions to be passed to FA. The approximate selection probabilities of solutions decrease exponentially as the ranks increase. For example, there is a 0.50 chance that a solution of first seven ranks is passed to FA. Similarly, 75% of solutions to be passed to FA are expected to be solutions of first 20 ranks.

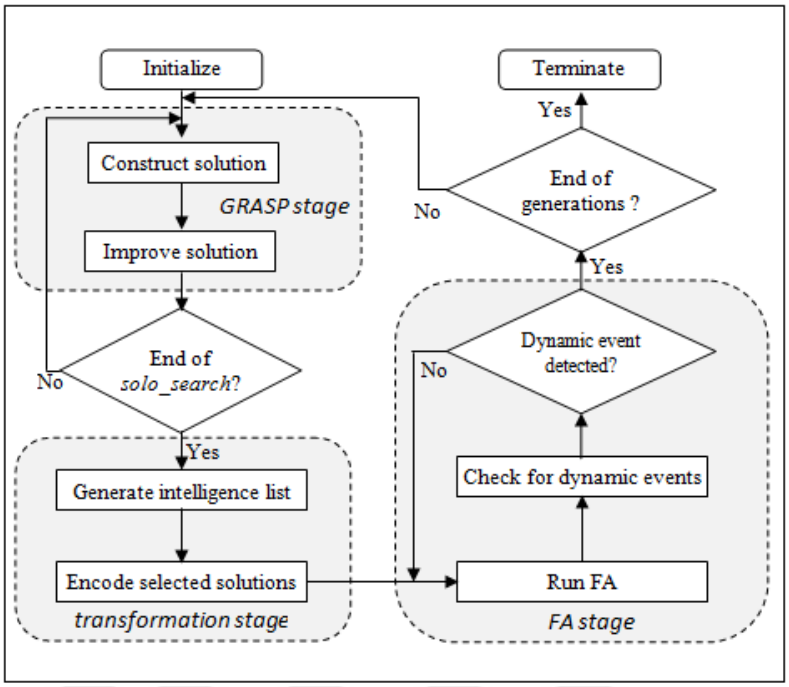


Figure 4.10 A flowchart for HybGrFA

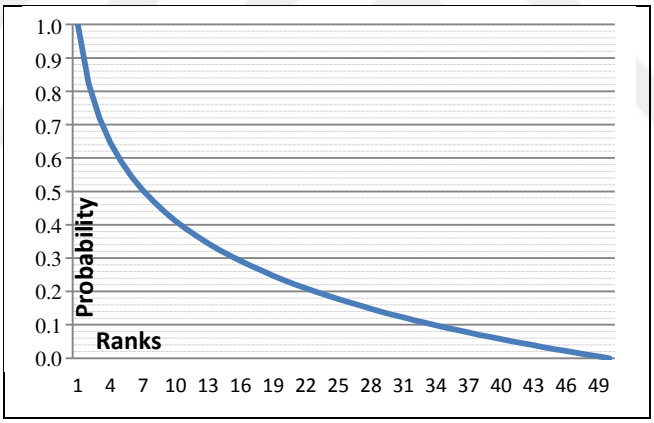


Figure 4.11 Sorted solutions and corresponding selection probabilities

4.3.6 Detecting Dynamic Changes and Responses

Similar to the previous chapter, random testing point technique is also used here for detecting dynamic changes. Because the entire problem related data is changed via dynamic events, the solution mapping and some corresponding indicators like profit, feasibility or degree of feasibility also changes. These indicators are evaluated at the beginning of each generation. If corresponding profit, feasibility or degree of feasibility of this random test point changes, it is concluded that a dynamic event is triggered.

In HybGrFA, as response to dynamic events, first GRASP is triggered, however, in GA, DE and FA, dynamic event handling technique is different. For these algorithms, a scheme similar to random immigrants is employed here. In random immigrants method, some random individuals are introduced to population within some periods or whenever the diversity of a population falls below a predefined threshold. This is a typical maintaining diversity based method. However, in the proposed approach, those random immigrants are allowed to be introduced only whenever a dynamic event is detected.

For this purpose, as 0.3, 0.5 and 0.7, three different diversity levels are determined here. In the highest diversity level, i.e. when this level is fixed to 0.7, subsequent to detection of a dynamic event, randomly selected 70% of the population is replaced with randomly generated individuals, where as the remaining 30% of the population is kept for the new environment. Additionally, the bits of the discarded items are also discarded from solution representations and bits for newly arrived items are appended to these representations. Finally, the priority values of these new bits are defined randomly between the minimum priority value and maximum priority value of the corresponding individual. Thus, each individual in the population relatively assesses the assignment of these new items into the knapsack. This is another advantage of using priority based encoding as solution representation method.

4.4 Computational Results

The experiments for the given algorithms are conducted on 50 consecutive dynamically generated environments, where dynamic events are triggered under low, medium, high and extremely high frequencies, which correspond to 5000, 2500, 1000 and 500 generations, respectively. All printed results are the averages of 30 independent runs. Optimum solutions of each environment is obtained using GAMS CPLEX solver. All tests are performed on a PC with hardware of 3.4 Ghz processor and 4 GB RAM.

4.4.1 Tuning the Parameters

It is obvious that the parameter *genMAX* is evaluated based on the number of dynamic environments and *frequency* of the instance. For example, if *frequency* is 500, because number of dynamic environments is fixed to 50, algorithm is terminated when *genMAX* reaches to 25000.

As one of the algorithms used in the present study, FA has three parameters, denoted by *pop_size*, $\beta_{(0)}$ and α, which correspond to the number of fireflies in a population, step size in movement of fireflies and randomness in movements (Chapter 3.3.1). The parameter *pop_size* might be increased arbitrarily, however, after achieving an adequate level; any further increment here does not contribute to solution quality. Moreover, required CPU increases exponentially because of the inner loops of FA, where pairwise comparisons are performed. On the contrary, if it is fixed to insufficiently smaller values, although algorithm performs faster, performance of the algorithm might deteriorate and possibly higher quality solutions are missed. Therefore, based on a tradeoff between contribution and required effort, the analysis, presented in Table 4.1 is performed.

Table 4.1 Offline error and CPU values for the corresponding *pop_size*.

pop_size.	offline error	CPU (sec)
25	262.89	35.19
50	247.98	72.64
90	156.40	172.07
100	144.36	180.21
110	152.33	219.42
130	146.64	308.16
150	142.35	451.89

According to the analysis presented in Table 4.1, as one can see, increment in *pop_size*, yields to a decrease in offline error to some extent. As a result, *pop_size* is fixed to 100 for all tests. For fair comparisons, *pop_size* values for both GA and DE are fixed to the same values. The other related parameters of DE and GA are adopted from Baykasoğu & Ozsoydan (2014).

The related analysis including the effects of $\beta_{(0)}$ and α is depicted in Figure 4.12.a. As the findings presented in this figure indicate, the best offline error is obtained when parameters $\beta_{(0)}$ and α are set to 0.35 and 0.9, respectively. Additionally, the worst offline error values are encountered while α equals to 0.1 regardless of $\beta_{(0)}$. As a result, $\beta_{(0)}$ and α are fixed to 0.35 and 0.9, respectively for all tests.

In Figure 4.12.b, another analysis is conducted to define appropriate levels of parameters Θ and *solo_search*. As mentioned earlier, *solo_search* is a stage, where GRASP performs a standalone search at the beginning of each changing environment. The parameter Θ defines the ratio of the solutions to be passed to FA and found by GRASP through *solo_search*. The number of solutions passed to FA is defined as Θ ×*solo_search*. Therefore, higher values of Θ and *solo_search* means a greater number of solutions passed to FA via GRASP.

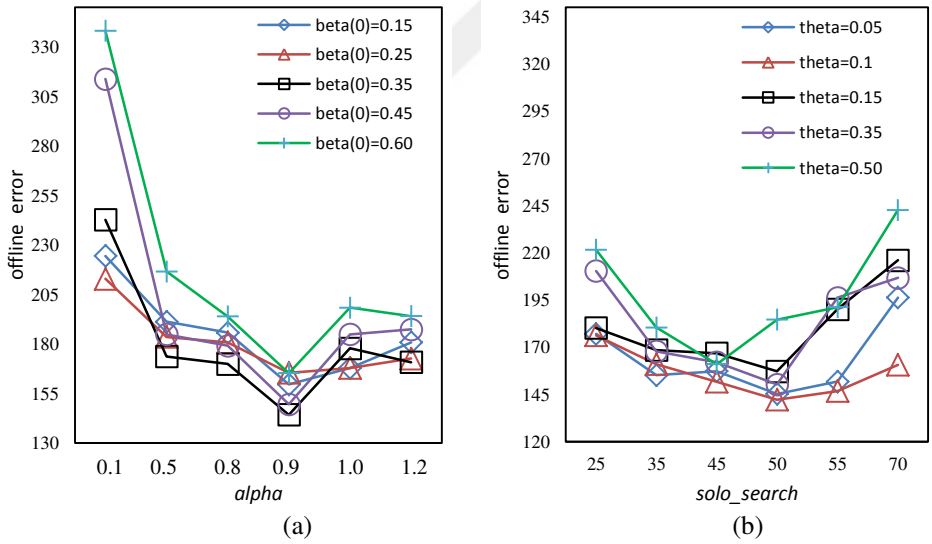


Figure 4.12 a)The effects of $\boldsymbol{\beta_{(0)}}$ and $\boldsymbol{\alpha}$ b)the effects of *solo_search* and Θ

As can be seen from Figure 4.12.b, the best offline error is obtained when parameters Θ and *solo_search* are fixed to 0.1 and 50, respectively. As the intuition says, whether the parameter *solo_search* increases, the probability of finding better solutions for GRASP increases as well. However, in such cases, the number solutions

passed to FA increases. This is the reason why the number of solutions passed to FA is evaluated based on the parameter *solo_search*.

In regard to the analysis depicted in Figure 4.12.b, increments in both Θ and *solo_search* do not contribute to the solution quality. This is possibly because of losing the diversity while introducing solutions from similar promising regions of search space. On the other hand, some leading solutions, passed to FA and combined with randomly introduced solutions have good effects in solution quality. As a result, in the present work, Θ and *solo_search* are fixed to 0.1 and 50, respectively.

Table 4.2 The effects of parameter $\boldsymbol{\eta}$

η	offline error
0.1	170.36
0.2	149.44
0.3	142.88
0.4	154.67
0.6	168.36
0.8	157.01
0.9	209.56

The final test focuses on finding an appropriate value for the parameter η, which regulates the balance between a random and deterministic search behaviour of GRASP throughout the solution construction stage. According to the analysis presented in Table 4.2, the best offline error is obtained when the parameter η is fixed to 0.3. Therefore, this value is used for the experimental study.

4.4.2 Performance Evaluation

Performance metrics, already introduced in Equations 2.8 and 2.10, are used here. These metrics are evaluated over generations. In Tables 4.3-4.4, the first and the second columns represent the frequency of change and level of diversity, respectively. The following columns denote the obtained offline error and in parenthesis standard error values for the corresponding algorithm.

According to these findings, it is clear that each of the algorithms achieves better results as the frequency increases. It is an expected outcome, because any algorithm has more chance to find higher quality solutions whether the period, which the

problem environment remains stationary, takes longer. However, one can see a surprising result if GA and DE are compared to FA in terms of different diversity levels of the same frequency. It is obvious that the performance of both GA and DE is not improved by introducing diversity, whereas increment in diversity level yields to positive effects for FA. Moreover, in most cases, increasing diversity level has minor positive effects for both GA and DE to some extent and in some cases further increasing the diversity level has negative effects on solution quality for these algorithms. This indeed reflects the results of losing whole or some part of the evolutionary information, gathered throughout generations. On the other hand, in FA, generally speaking, increasing diversity level from 0.3 to 0.5 or from 0.5 to 0.7, clearly increases the performance of the algorithm. As a result, it can be concluded that recovery of GA and DE can be considered as slower than the recovery of FA and this affects the overall performance represented by offline error values.

Another surprising result is apparent in comparison of GRASP and HybGrFA. Although HybGrFA is a further extended modification of GRASP, it outperforms GRASP only in the extremely high frequency experiment presented in Table 4.3. Additionally, while increasing frequency from 2500 to 5000 has an approximately 19% contribution in GRASP, it yields to only 0.85% improvement in HybGrFA. The reaction of HybGrFA to changes in frequency is similar to the reactions of GA and DE. However, change in frequency, has more apparent effects on solution quality for both GRASP and FA. Finally, it can be concluded that, although HybGrFA achieves better standard error values according to the results printed in Table 4.4, as an overall assessment, GRASP with its current form is found as a more promising solution approach throughout the experiments conducted here.

Convergence of current error and offline error values of each algorithm for the corresponding frequency levels are illustrated in Figures 4.13 and 4.14, respectively. For GA, DE and FA, all plotted values are the mean over the diversity levels of the corresponding frequency level. As one can see from Figure 4.13.a, GRASP and HygGrFA can be considered as two competitive algorithms however, it is clear that the recovery of HybGrFA is better than the recovery of GRASP. Because in GRASP, the current error of the algorithm increases just after the dynamic event, whereas in

HybGrFA, GRASP is triggered first and it introduces some higher quality solution to FA. Thus initial current error values of HybGrFA just after a dynamic event is better than the initial current error values GRASP. However, this feature alone is not enough for a better overall performance. As can be seen from the rest of frequency levels (Figure 4.13.b-d), GRASP converges to better solutions, while HybGrFA cannot sufficiently improve the initial solutions.

Table 4.3 Offline error (standard error) values according to Equation 2.8

frequency	*diversity level*	GA	DE	FA	GRASP	HybGrFA
500	0.3	3204.01 (4.03)	2208.63 (2.19)	1396.76(16.46)		
	0.5	3182.64 (3.65)	2206.55(2.23)	757.52 (5.75)	185.25(0.98)	163.80 (1.26)
	0.7	3200.65 (2.82)	2214.20(2.01)	441.17 (4.26)		
1000	0.3	3058.49 (3.31)	1905.64 (2.24)	1273.21 (19.83)		
	0.5	3052.05 (3.64)	1904.98 (2.15)	684.38 (7.89)	154.71(0.81)	160.60 (1.59)
	0.7	3059.92 (3.36)	1906.10 (2.11)	360.26 (4.45)		
2500	0.3	2888.18 (3.77)	1530.90 (2.32)	1210.00 (18.16)		
	0.5	2892.08 (2.98)	1536.07 (1.56)	607.75 (8.94)	116.28(0.59)	159.88 (1.62)
	0.7	2892.03 (2.97)	1532.02 (2.18)	308.65 (3.88)		
5000	0.3	2766.09 (3.05)	1267.42 (1.66)	1227.50 (13.02)		
	0.5	2767.82 (2.87)	1270.47(1.78)	590.86 (7.93)	93.51(0.61)	157.46 (2.13)
	0.7	2770.37 (2.69)	1268.57 (1.72)	293.82 (3.57)		

Table 4.4 Offline error (standard error) values according to Equation 2.10

frequency	*diversity level*	GA	DE	FA	GRASP	HybGrFA
500	0.3	2989.00 (37.01)	1773.50(22.91)	1198.22(42.95)		
	0.5	2973.53 (36.69)	1760.26(22.51)	571.14 (34.83)	134.05 (7.34)	157.63 (5.85)
	0.7	2985.60 (35.58)	1773.29(23.51)	272.32 (20.40)		
1000	0.3	2857.29 (32.87)	1485.36 (20.49)	1171.29 (44.06)		
	0.5	2858.14 (35.53)	1487.69 (20.25)	591.63 (33.08)	110.83 (7.01)	156.70 (5.54)
	0.7	2866.88 (33.96)	1482.99 (20.13)	275.38 (21.14)		
2500	0.3	2709.54 (33.79)	1150.32 (16.60)	1168.78 (44.77)		
	0.5	2715.77 (33.47)	1142.22 (17.05)	569.94 (32.46)	80.66 (6.32)	157.95 (5.72)
	0.7	2711.31 (33.81)	1135.57 (16.19)	274.34 (22.10)		
5000	0.3	2598.34 (33.32)	911.56 (13.20)	1206.75 (43.29)		
	0.5	2601.25 (32.49)	912.74 (13.41)	571.67 (32.16)	61.96 (6.48)	156.61 (5.82)
	0.7	2599.57 (33.38)	911.56 (13.95)	276.12 (22.40)		

A similar result is apparent in Figure 4.14. As one can see from Figure 4.14.a-b, the offline error lines of GRASP and HybGrFA are approximately overlapping. This shows that for extremely high and high frequency levels, these algorithms might be considered as two competitive algorithms, however for the rest of the frequencies, GRASP outperforms the rest of the algorithms again.

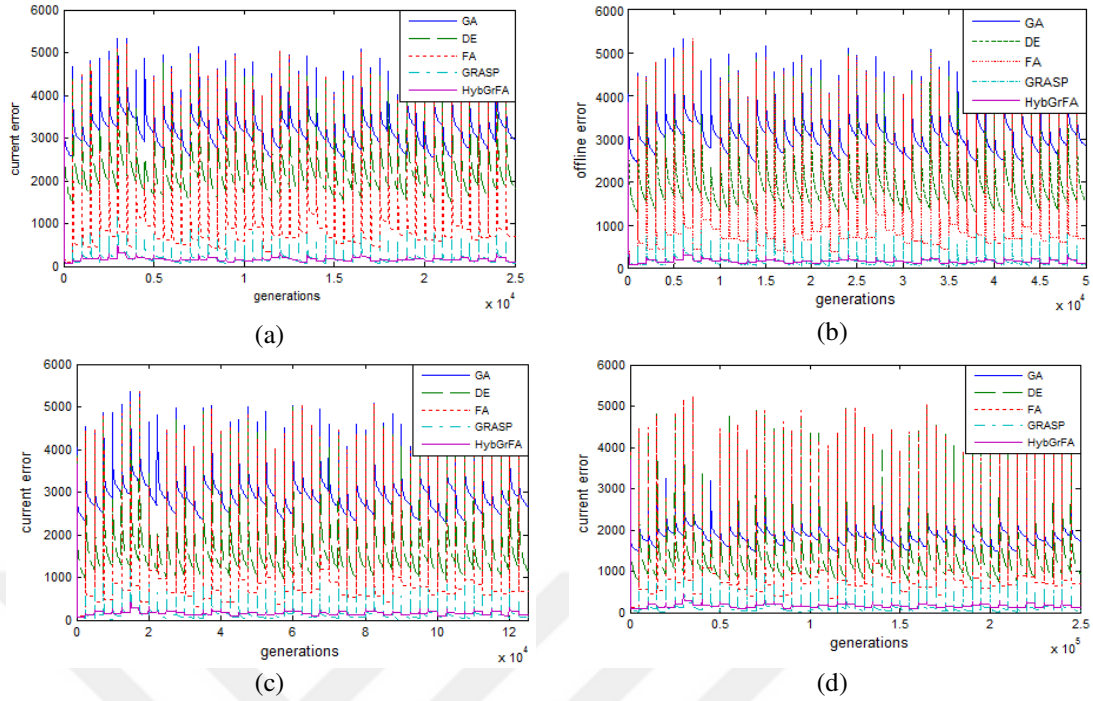


Figure 4.13 Current error values of the algorithms under a) 500 frequency, b)1000 frequency, c) 2500 frequency, d) 5000 frequency

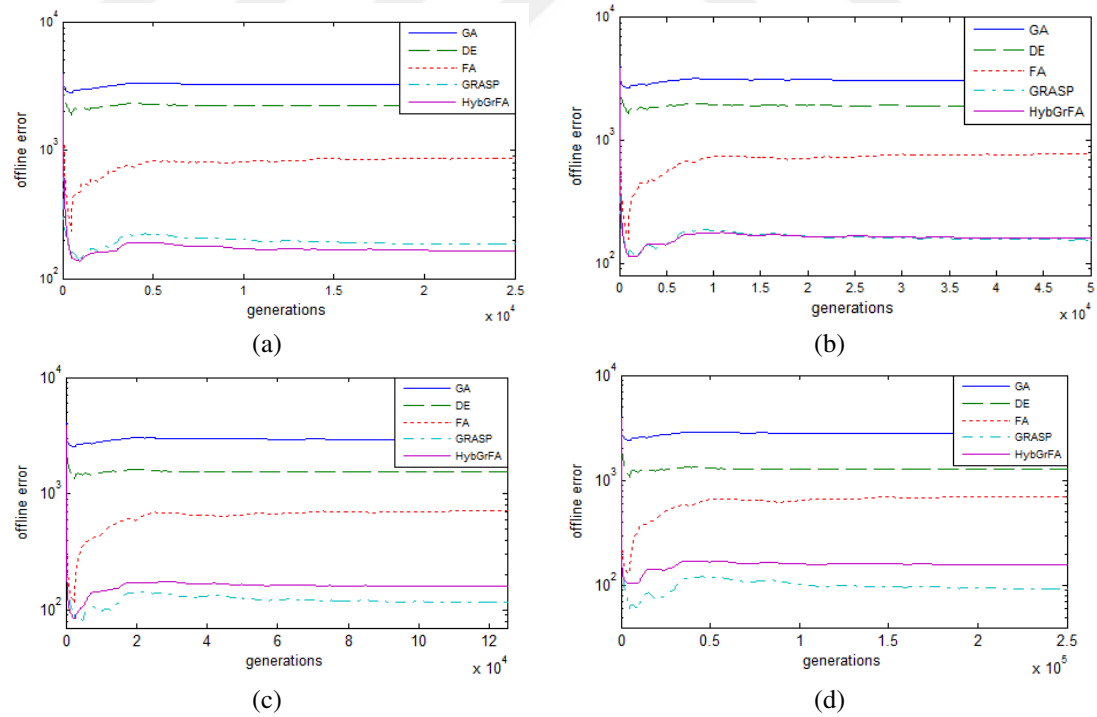


Figure 4.14 Offline error values of the algorithms under a) 500 frequency, b)1000 frequency, c) 2500 frequency, d) 5000 frequency

4.4.3 Statistical Verification

Here, a statistical verification is performed to demonstrate what intuition suggests previously. For this purpose, an initial examination is conducted to see whether a significant difference among algorithms exists. Therefore, Friedman Test (Friedman, 1937; Friedman, 1940) which is particularly applicable to multiple classifiers is performed first.

Using the data sets of 30 runs, F_F values for 500, 1000, 2500 and 5000 frequency levels are obtained as 1582.11, 595.40,+∞ and +∞, respectively (due to division by zero). With 5 algorithms and 30 data entries (final offline error values of each run for the corresponding algorithm), F_F is distributed according to F distribution with (5-1)(30-1)=116 degrees of freedom. The critical value of $F_{0.05,4,116,}$ for α=0.05 is approximately 2.45. As a result, the assumption under null-hypothesis (H_o) imposing that each algorithm achieves equal average ranks is rejected for each of the frequency levels. Therefore, we can conclude that a significant difference from mean rank exists, which statistically proves that performances of algorithms significantly differ from each other. However, a further analysis is required to prove the superiority of any algorithm. Therefore, in order to demonstrate a possible critical difference (CD) between any pairs of algorithms, the Bonferroni-Dunn test (Dunn, 1961) is conducted next.

In this test, first, algorithms are ranked w.r.t. to their final offline error values for the corresponding frequency level. The reel differences (RD) between averages of the ranks are illustrated in Table 4.5. In Bonferroni-Dunn test, $q_{0.10}$ is equal to 2.241. Thus, CD is calculated as 0.915. If RD between any pairs of compared algorithms is found as greater than this CD value, it is concluded that a critical difference between the compared algorithms exists.

According to the results printed in Table 4.5, it is clear that the findings are parallel to the previously found outcomes. For extremely high and high frequency levels, where the frequency is fixed to 500 and 1000, respectively, a statically verified critical difference does not exist between HybGrFA and GRASP under the given parameters of these tests. However, for the rest of the tests, a CD exists

between each pair of the compared algorithms. Therefore, it can be concluded that the constructive and multi-start search strategy for DCOPs has an apparent advantage in comparison to EAs and population based algorithms for the experiments conducted here under the given parameters and assumptions. Additionally, one can note that canonical versions of GA, DE and FA (except the move in FA, which uses Equation 3.4) are employed here. It is clear that more sophisticated extensions of these algorithms might perform better and yield to different results.

Table 4.5 Statistical results for Bonferroni-Dunn test

frequency	GRASP-HybGrFA	GRASP-FA	GRASP-DE	GRASP-GA	HybGrFA-FA	HybGrFA-DE	HybGrFA-GA	FA-DE	FA-GA	DE-GA
500	*RD*=0.80 <*CD*	*RD*=1.10 >*CD*	*RD*=2.10 > CD	*RD*=3.10 > CD	*RD*=1.90 > CD	*RD*=2.90 > CD	*RD*=3.90 > CD	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=1.00 > CD
1000	*RD*=0.27 < CD	*RD*=1.63 > CD	*RD*=2.63 > CD	*RD*=3.63 > CD	*RD*=1.37 > CD	*RD*=2.37 > CD	*RD*=3.37 > CD	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=1.00 > CD
2500	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=3.00 > CD	*RD*=4.00 > CD	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=3.00 > CD	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=1.00 > CD
5000	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=3.00 > CD	*RD*=4.00 > CD	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=3.00 > CD	*RD*=1.00 > CD	*RD*=2.00 > CD	*RD*=1.00 > CD

4.5 Chapter Conclusion

The focus of this chapter is proposing the idea of making use of a constructive and multi-start search strategy for combinatorial dynamic optimization problems. Furthermore, an algorithm referred as HybGrFA, which hybridizes a constructive and a population based algorithm is introduced.

The proposed hybrid metaheuristic HybGrFA, combines the intrinsic features of GRASP and Firefly Algorithm. GRASP is a constructive and discrete algorithm, whereas Firefly Algorithm is particularly designed for continuous search spaces, which makes simultaneous use of these algorithms challenging. For this reason, thorough presentations of the developed hybridizing methodologies along with specialized cooperative procedures are provided within this chapter.

For numerical demonstrations, extensive performance analysis is performed in regard to specific performance measures, commonly used in dynamic optimization community. Due to lacking benchmarking data and similar approaches, GRASP, HybGrFA, Firefly Algorithm, Differential Evolution Algorithm and Genetic

Algorithm are compared to each other. As demonstrated by the numerical analysis and graphical illustrations, constructive and multi-start search strategy is found as an effective and a promising algorithm for combinatorial dynamic optimization problems, where both dimensional and non-dimensional changes exist. This approach is expected to contribute to the existing dynamic optimization literature by the features listed below:

- ✓ Dimensional changes and time varying parameters are effectively handled by constructive and multi-start strategy based approaches without making use of further adaptation methods, frequently used in dynamic optimization community.
- ✓ Regardless of the frequency, constructive strategy is able to monitor the promising regions of diversified parts of search space.
- ✓ An additional memory to keep a record of the promising solutions is not required here, because, in its nature, current solutions obtained in GRASP are not dependent to previously found solutions. In other words, GRASP is looks into only the current available data. In this manner, its behavior can be considered as "what is in the menu" based approach.

The proposed strategy might be implemented on different combinatorial optimization problems such as scheduling, routing problems, etc., which can be scheduled as a future work.

CHAPTER FIVE
ONLINE AND DYNAMIC SCHEDULING OF PARALLEL MACHINES USING A CONSTRUCTIVE AND MULTI-START STRATEGY: A CASE STUDY IN A FASTERNERS MANUFACTURING COMPANY

This chapter presents a case study, where a multi-start and constructive algorithm is proposed for online and dynamic scheduling of parallel heat treatment furnaces in a manufacturing company, which is particularly operating in automotive industry. The mentioned problem involves, release times, eligibility constraints, due dates, sequence-dependent setup times due to heating up or cooling down the furnaces, breakdowns and maintenance periods of those furnaces. Dynamic events such as arrivals of rush orders, cancellations of already handled jobs, changes in due dates or in lot sizes exist in nature of the handled problem. The proposed dynamic and self-directed scheduling module of the heat treatment furnaces is embedded to the ERP system of the company. Based on scientific and modern optimization techniques, the current system allows, dynamic and online scheduling of the heat treatment furnaces, online generation of the Gantt Charts for managers of the related departments, online generation of work orders and digitalized performance analysis reports of the corresponding schedules.

5.1 Chapter Introduction

Scheduling problems (Cheng & Sin, 1990; Lee, Fox & Watkins, 1993; Ovacik & Uzsoy, 1995b; Pinedo, 2012) have attracted a notable attention of both researchers and practitioners. As an extension of this problem class, parallel machine scheduling problem (PMSP), which is one of the most widely practiced problems, has numerous practical applications in industry, commercial manufacturing systems and expert systems.

As most of the combinatorial problems, PMSPs are known to be NP-hard (Ullman, 1975; Pinedo, 2012), which in turn brings a considerable challenge to both

practitioners in real-life applications and researchers from academic community. In the early era of this field, a vast variety of PMSP papers is reported in the literature. As one of the first attempts, Cheng & Sin (1990) classify PMSP into three categories namely, identical, uniform and unrelated parallel machine problems. However, PMSPs have further attributes to be considered for classification. As one of them, setup operation (time and/or cost), which has for long been considered negligible and hence ignored, or considered as part of the processing time (Allahverdi, Gupta & Aldowaisan, 1999; Allahverdi, Ng, Cheng & Kovalyov, 2008), has a great impact on the efficiency of the generated schedules. Particularly if those setup operations are dependent to the sequence of two consecutive jobs, the significance of the problem increases as well. Although in recent studies and advances sequence-dependent setup time is given special attention, this piece of PMSPs is still lacking in comparison to other extensions of this problem. A comprehensive presentation is reported in Armentano & de Franca Filho (2007). Some other related surveys are also presented in Allahverdi et al. (1999), Pfund, Fowler & Gupta (2004) and Allahverdi et al. (2008).

In one of the recent efforts, Ying & Cheng (2010) propose an iterated greedy search heuristic for a PMSP with sequence-dependent setup times and release times. One interesting work is addressed by Gokhale & Mathirajan (2012), where a scheduling problem with identical parallel machines, eligibility restrictions, release times and sequence-dependent setup times exist in the mentioned problem, encountered in an automobile gear manufacturing. An extension of PMSP with machine preference is addressed in Huang & Liao (2012). Edis & Ozkarahan (2011) report a similar problem. Ruiz-Torres, Paletta & Pérez (2013) introduce deteriorating effects of jobs on machines, which affect process times in advance. In the same year, Costa, Cappadonna & Fichera (2013) propose a genetic algorithm for labor allocation in parallel unrelated machines with sequence-dependent setup times. A survey focusing on online scheduling with machine eligibility restrictions is reported by Lee, Leung & Pinedo (2013). Other surveys related to real-time scheduling are presented in Davis & Burns (2011) and Buttazzo, Bertogna & Yao (2013). Some real-life applications adopting GRASP in real-time scheduling problems are given in Argüello et al. (1997) and Khadidja et al. (2012). Lin & Hsieh (2014) employ a

heuristic and an iterated hybrid metaheuristic for minimization of total weighted tardiness in PMSP, where sequence and machine-dependent setup times and ready times exist. In the same year, Yilmaz Eroglu, Ozmutlu & Ozmutlu (2014), propose a genetic algorithm for unrelated PMSP with sequence-dependent setup times and with the objective of makespan minimization. Baykasoglu & Ozsoydan (2015b) present another work proposing to employ adaptive neuro fuzzy inference systems to determine sequence dependent setup times for vast variety of products. Finally, Xiao, Yang, Zhang, Zheng & Gupta (2015) present an interesting application in a semiconductor factory, which takes the lot-sizing and sequence-dependent setup times, time windows, machine eligibility and preference constraints, into account.

In the present case study, a constructive and multi-start strategy is proposed for solving a special case of parallel machine scheduling problem with release times, sequence-dependent setup times, eligibility constraints, due dates and machine breakdowns. The focused problem includes both dimensional and non-dimensional dynamic events, where in dimensional changes arrivals/cancellations of orders or machine breakdowns/maintenance occur and non-dimensional changes correspond to changes in problem data like changes in due dates, lots sizes etc.

The proposed approach has several merits particularly in real-life applications. First, it only deals with the revealed part of a problem environment, so any dynamic event detection mechanism is not necessarily required here. In other words, at the beginning of each generation, it takes a picture of the system and runs only with the currently available data. As a result, in the proposed algorithm, dependency to previously found solutions is eliminated in contrast to numerous other dynamic scheduling algorithms, where particularly evolutionary of population based techniques are employed. Another advantage of this approach is that it guarantees feasible and applicable solutions in short period. This is a crucial necessity for real-life manufacturing systems.

The rest of this chapter is organized as follows: Section 5.2 is devoted to clarifying the handled problem. The details of the case study and proposed solution

approach are presented in Section 5.3. Implementation in the firm and concluding remarks are given in Section 5.4 and Section 5.5, respectively.

5.2 Problem Statement

The mentioned problem can simply be summarized as online scheduling of parallel heat treatment furnaces under dynamic operating conditions. The aim of heat treatment is to provide the required endurance, durability, strength and resistance to the fasteners.

The focused problem here includes a number inherent feature. Firstly, the mentioned problem case involves previously unknown and hard-to-predict future events such as arrivals of rush orders, cancellations of already handled jobs and changes in due dates or lot sizes. Existence of such factors makes the problem dynamic. It is worth stressing that numerous published works adopt the word dynamic whether only release times exist in the problem. However, only existence of release times does not make a scheduling problem dynamic. Since the release times are known beforehand, one can just schedule any job just when it is released. In other words, the release times of the jobs are already revealed for a decision maker and the problem environment remains stationary throughout the processing the scheduled jobs. However, in the present case study, although some previously known to be released jobs exist as in traditional release time restrictions, additionally, there are rush orders, whose arrivals are previously unknown and remain unrevealed until they arrive at the system. Likewise, previously scheduled jobs might be cancelled or postponed with urgent decisions and parameters like or due dates, lot sizes, etc. might change during the processing of the scheduled jobs. In the present case study, only such cases are referred as dynamism factors. The features of the focused problem are listed in the following.

5.2.1 Dynamism Factors

As mentioned above, two types of dynamism factors simultaneously exist in the current problem. In the former one (dimensional), arrivals/cancellations of jobs and periodic maintenance/breakdowns of the furnaces yield to changes in problem

domain. In this type of dynamism factor, the sizes of the sets, which are used to define the problem, unpredictably change. In the latter one (non-dimensional), changes in due dates, in priorities of the jobs and lot sizes only affect the related problem data.

5.2.2 Sequence-dependent Setup Times

Consecutive jobs, processed in heat treatment stage, require different temperature levels. For this reason, a heat treatment furnace might need to be heated up or cooled down between consecutive jobs. However, heating up and cooling down the furnaces differ considerably. Heating the furnaces up from 400^0C to 600^0C is not the same task as cooling them down from 600^0C to 400^0C. Thus, assuming that $i \neq j$, elapsed time while preparing a furnace from job i to job j differs from the elapsed time between jobs j and i. Those elapsed times correspond to sequence-dependent setup times and setup only occurs while heating up or cooling down period of a heat treatment furnace, which means that a setup can be initialized whether the forthcoming job is released or not. This reveals an interesting practical application clarified in the following.

Let's assume that job j is already released ($r_j \leq c_{j-1}$). Thus, as depicted in Figure 5.1.a, job j can be started at the time str_j, which can be calculated as the sum of c_{j-1} and $s_{j-1,j}$, where, r_j, str_j, c_{j-1} and $s_{j-1,j}$ represent release and starting time of job j, completion time of job j-1 and setup between jobs j-1 and j, respectively. Additionally, if it is known that job j is released during the setup $s_{j-1,j}$, which means that $r_j \leq c_{j-1} + s_{j-1,j}$ is satisfied, then the schedule presented in Figure 5.1.a can still be applied. However, if job j is not yet released and if $r_j > (c_{j-1} + s_{j-1,j})$ condition is satisfied, for the focused problem, str_j can be equalized directly to r_j and setup between jobs i and j can be initialized although job j is not currently released (Figure 5.1.b). This is one of the interesting issues encountered in the present study. Furthermore, as one can notice that $s_{j-1,j}$ might seem like a slack block, which can be slide and be placed between c_{j-1} and str_j in Figure 5.1.b. However, this is not the case in practice. Although the job j-1 is completed, the

temperature of the furnace is kept at the same level with the required temperature of job *j*-1 and setup is placed just before the starting time of job *j*. On the other hand, in the common assumption (Figure 5.1.c), it is assumed that job *j* needs to be physically released in order to attach it to its scheduled machine and to perform its specific configurations. However, in the present case study, such a necessity is not required because setup occurs while only heating up or cooling down a furnace. In practice, this circumstance makes a significant difference.

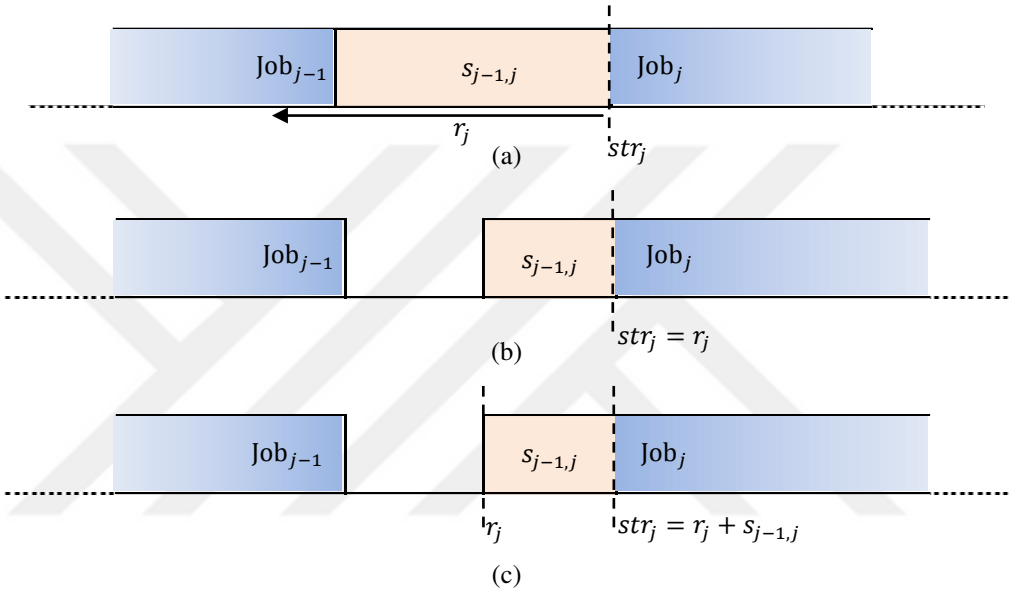


Figure 5.1 a) job *j* is already released or will be released during the setup b) job *j* is not yet released and $\boldsymbol{r_j > (c_{j-1} + s_{j-1,j})}$ for the focused problem c) job *j* is not yet released for the common assumption

5.2.3 Eligibility Constraints

Although each of the heat treatment furnaces are identical, according to some production limitations arising from the expensiveness of some specific components like zinc-phosphate tanks, only some of those furnaces have full configuration. Thus, some of the jobs can only be processed on some predefined furnaces.

5.2.4 Implicit-explicit Priorities

If the operational conditions run in ordinary course, each of the jobs has identical priorities, which means that any job is not more important than any other job.

However, under some particular conditions like in high demand periods, in order to satisfy the tight agreements with customers, some of the jobs (orders) are assigned with implicit or explicit priorities. Implicit priorities indeed correspond to weights, which show the importance of the jobs, and those jobs with higher weights are attempted to be scheduled at the forefront positions of each production stage as much as possible. Alternatively, an explicit priority can be assigned to a job, which shows the urgency of that job. In practice, such jobs are referred as the *jobs with red flags* and the jobs with red flags are directly assigned as the first jobs in all queues of each production stages. In other words, the jobs with red flags are scheduled to positions, where they can be completed at an earliest possible date.

5.2.5 Release Times

According to the manufacturing process of the firm, the former process stage prior to heat treatment is the cold forming stage. The cold forming completion time of a job is indeed the release time of that job for the heat treatment process.

5.2.6 Due Dates

Each of the jobs recorded in ERP system of the firm have different due dates. Those dates are arranged in regard to individual customer agreements. Although too tight due dates do not exist, particularly in high demand periods, they might become the first priority issue of concerns.

5.2.7 Maintenance/breakdown of the Heat Treatment Furnaces

Although breakdown is not a frequently encountered situation in the heat treatment process, periodic maintenance of the furnaces is an important factor while generating schedules. Throughout the maintenance period, a furnace cannot be used for approximately one week.

5.2.8 Process Times

There exist minor and ignorable differences between heat treatment furnaces, arising particularly from manufacturing date of those furnaces. According to the

decisions taken in meetings with the authorized personnel in the firm, the heat treatment furnaces are assumed as identical machines except the eligibility constraints. As a result, each job has equal process times on each of the furnaces.

5.2.9 The Objective

Heat treatment department of the firm aims to reduce the ineffectively wasted times in the furnaces, which is indeed a result of unnecessarily long setup times, arising from disorganized schedules, which are prepared according to intuition and experience of the authorized personnel in production planning department. If inefficiently spent time is reduced, energy consumption in furnaces and maximum completion time of a production schedule is also reduced. Therefore, the first priority is minimization of the maximum completion time (C_{max}) of a schedule. Alternatively, if any two schedules have equal C_{max} values, the one with a lower maximum tardiness value is chosen.

All of the attributes of the focused problem are stated above in details. As one can see from those definitions, it is hard to define an appropriate problem class for the related problem. Considering the standard $\alpha \mid \beta \mid \gamma$ notation of Graham, Lawler, Lenstra & Rinnooy (1979), the problem falls into $P \mid s_{jk}, r_j, M_j, brkdwn \mid C_{max}, L_{max}$. It shall be noted that two objectives are simultaneously denoted in γ field of this triplet. However, it is not a bi-objective application of C_{max} and L_{max}. As stated above, the first priority is the minimization of C_{max}. As secondarily, among from the schedules with equal C_{max} values, the schedule with the minimum L_{max} value is chosen.

5.3 Diagnosis and Treatment

In this section, along with general production flow process (Figure 5.2), brief information is provided about the manufacturing company, where the proposed project is held. Next, the details of the problem, including particularly heat treatment department related components are presented.

5.3.1 A Brief Introduction of the Project Partner and Its General Production Flow

The project partner firm, where the proposed algorithm is implemented, was founded in Izmir/Turkey in 1973. It is currently operating on its new plant with area of 58.000m^2. The firm has a wide range of quality certifications including TS 16949:2002, ISO 14001, ISO 9001:2000, Q1, EN 14399:1, ISO 18001. The company is particularly operating in automotive and white goods industry and it is currently in supplier position for about 30 brands, including worldwide, well-known, leading car brands. As an average, the firm produces 50.000 tons of fasteners (screw, bolt, ring nut, bell crank, suspension bolt, rivet) annually.

A general production flow diagram applied in the firm is presented in Figure 5.2. According to this figure, raw material, kept in coils, is first exposed to annealing process. Next, surface cleaning procedures are performed in cleaning baths and zinc-phosphate baths in sequence. At this stage, chemical reactions remove dirt, rust and mildew, cumulated on the raw material due to some outer factors like, air, humidity and temperature. Afterwards, the cleansed raw material is shaped and is formed at the cold forming stage. At this stage, dimensions like length, width and depth of the cleansed material are adjusted. Raw material carried as a single coil body is cut and is pressed to obtain thousands of smaller pieces of fasteners including screw, bolt, ring nut, bell crank, suspension bolt and rivet. Cold forming (Figure 5.3) is an important stage for the heat treatment process. Because completion time for a job at the cold forming stage is indeed the release time of that job for the heat treatment process.

After cold forming stage, the prepared fasteners are carried to the heat treatment process via either forklifts or pallet trucks. In the firm, there are currently five parallel furnaces, each operated by an individual operator via RFID supported kiosk machines (Figure 5.4.a), connected to ERP system of the firm. Each of the furnaces has one individual kiosk machine, which is also used for communication with the production planning department. By using these machines, the production planning department sends the schedules as electronically prepared work orders to the operators, who are responsible for the furnaces they are assigned. Additionally, the

operators also regulate the temperature of the furnaces according to the work orders coming from the production planning department.

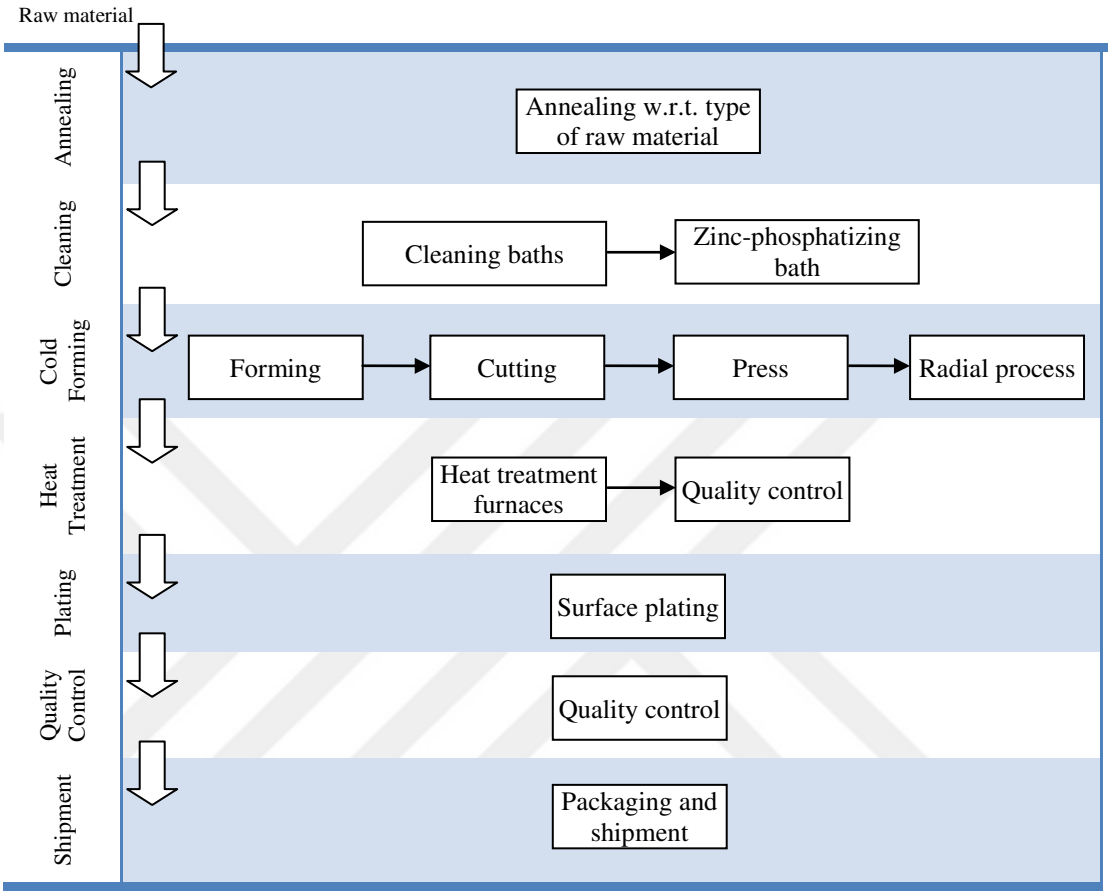


Figure 5.2 General production flow diagram of the firm

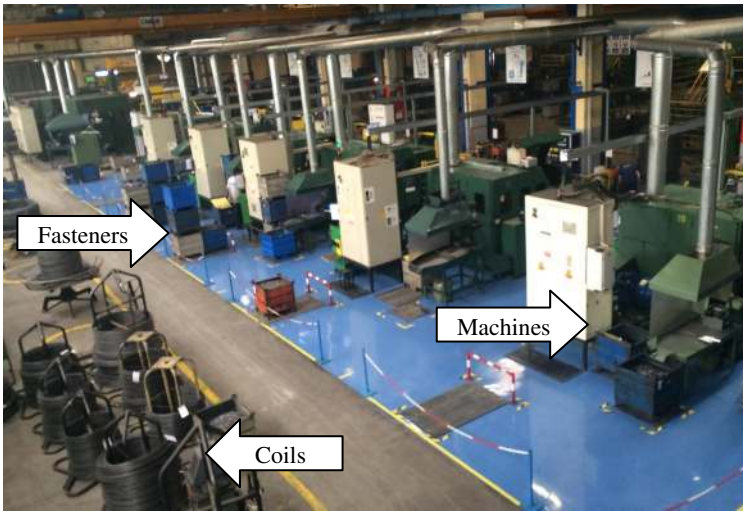
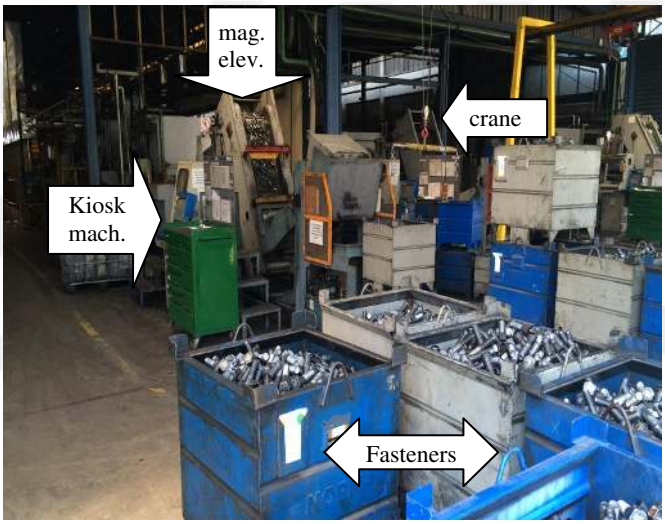
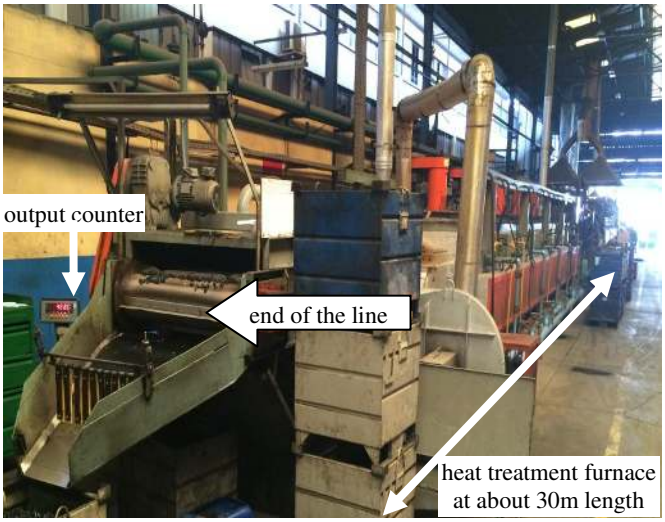


Figure 5.3 A snapshot of the cold forming stage

(a)



(b)



(c)

Figure 5.4 a)Kiosk machines b)a snapshot while loading a heat treatment furnace c)whole line view

Before loading a furnace, first, the corresponding lot is introduced to the system by the operators using RFID. Thus, any authorized personnel can follow that job on the ERP system. Next, the fasteners, carried in metal boxes (jobs) are placed onto an electromagnetic elevator via a crane, carrying those boxes (Figure 5.4.b) and thus, heat treatment process is initialized. Like chemical reaction tanks, shocking and oiling baths, there are several other components of a heat treatment furnace but they are out of the scope of this study. Finally, a full view of a heat treatment furnace is presented in Figure 5.4.c. As one can see from this figure, the completed (heat treated) fasteners are collected in metal boxes again. By the help of the output counter, which is already connected the ERP system of the firm, any authorized employee or manager can follow the outputs of the furnaces as well as the completion rate of any order online at any time.

After heat treatment process, a local quality control is performed. Random sampling is used for this control. Randomly chosen fasteners are exposed to size and durability tests. If a significant statistical data for rejecting a job is obtained, then that party is dismissed and a new order is defined from scratch. Because, identical conditions in furnaces for re-working cannot be maintained. This is the reason why the heat treatment process is an expensive and fragile stage. Finally, it is worth stressing that, according to the preliminary analysis performed at each stage of the production flow, heat treatment process is found as the bottleneck stage, which means that a possible improvement in scheduling of the furnaces might yield to apparent gaining for whole system.

Surface plating stage follows the heat treatment process. However, not all of the jobs are exposed to surface plating. Only some special orders are surface plated and this is one of the expensive stages in production. The aim here is to provide a further enhanced durability to the fasteners.

The final quality control is performed just before packaging and shipment. If a party cannot pass the quality tests, that party is discarded, although it is not a frequently encountered situation. Finally, the completed orders are packaged and are shipped to customers.

5.3.2 The Proposed Algorithm

The proposed approach should be parameter-free or least it should have fewer parameters to fix because it will be practically used in a real-life manufacturing system. In this respect, Greedy Randomized Adaptive Search Procedure (GRASP) of Feo & Resende (1995) is employed here. The details of GRASP are already provided in Chapter 4.3.1. Therefore, only the modification of this algorithm to the mentioned problem is presented here.

5.3.2.1 Construction Stage

As denoted by $\alpha \in [0,1]$, this stage has only one parameter, used for determining the length of RCL. In the proposed approach, in order to evaluate the greedy values of jobs, a simplified extension of the MATCS function, proposed by Kim & Shin (2003), which is a further modification of the ATCS, introduced by Lee, Bhaskaran & Pinedo (1997), is employed. It shall be noted that these methods are first introduced for scheduling problems including unrelated machines, however, there are identical machines in the focused problem and accordingly, in this extension presented in Equations 5.1-5.3, process times and sequence-dependent setup times are assumed as identical for all machines. As one can note, the mentioned function takes ready times, due dates, processing times, sequence-dependent setup times and weights into account.

$$I_{(j,t)} = \frac{w_j}{p_j} \exp\left[-\frac{\max\left(d_j - p_j - t, 0\right)}{k_1 \bar{p}}\right] \tag{5.1}$$

$$\exp\left[-\frac{s^*_{ij}}{k_2 \bar{s}}\right]$$

$$s^*_{ij} = \begin{cases} s_{ij} & \text{if } r_j \le t \\ r_j - t + s_{ij} & \text{otherwise} \end{cases} \tag{5.2}$$

where $I_{(j,t),}$, w_j, p_j, d_j, r_j, s_{ij} and t represent the greedy index of the *j*th job at time t, weight, process time, due date and release time of the *j*th job, sequence-dependent setup time between the *i*th and *j*th jobs, and the current time in horizon, respectively.

k_1 and k_2 are look-ahead parameters, which can be determined by some specific evaluations, as suggested in Lee et al. (1997).

As can clearly be seen from this function that the former exponential term gives higher priorities to jobs, which are closer to their due dates (or tardy), and the latter one prioritizes ready jobs, which have relatively smaller setup times.

Once the, greedy indices are evaluated as formulated in Equations 5.1-5.2, in order to obtain the final greedy values, they are normalized using the formulation given in Equation 5.3, where $g_{(j,t)}$ denote the greedy value of the *j*th job at time t. Next, RCL for time t is generated including the jobs, whose greedy values are in $g_{(j,t)} \in \left[\min_j g_{(j,t)} + (1-\alpha)\left(\max_j g_{(j,t)} - \min_j g_{(j,t)}\right),\ \max_j g_{(j,t)}\right]$ so that when α is fixed to unity, a pure random selection is performed by the algorithm. On the other hand, setting α to zero associates the algorithm with a pure greedy selection.

$$g_{(j,t)} = {I_{(j,t)}}\Big/{\sum_j I_{(j,t)}} \tag{5.3}$$

A solution in a single GRASP generation is constructed as follows. At the beginning, the first freed machine is found ignoring the ones, which are in maintenance period or transiently unavailable. Availabilities of the furnaces are accessible on the ERP system of the firm. Thus, algorithm can easily detect which furnaces are currently available. Next, among from all unscheduled jobs, the ones assignable to that machine are filtered. If none of the jobs is assignable to that machine, the next freed machine is found and same filtering is performed again. Thus, eligibility constraints as well as the availability conditions of the furnaces are handled without any extra effort. After obtaining a positive number of jobs, assignable to any found machine, following Equations 5.1-5.3, RCL is generated considering those candidate jobs. Finally, among from the jobs contained in RCL, one random job is appended to the end of the schedule of the found machine, considering the special case, previously mentioned in Figure 5.1. The assigned job is removed from the list of unscheduled jobs and same steps are followed until all jobs are scheduled to appropriate heat treatment furnaces. This procedure is repeated until

the termination criterion of GRASP is satisfied. Achieving a maximum number of generations as well as a user supplied maximum elapsed time is used here as two termination criteria. If one them is achieved, GRASP is terminated and the best-found solution is applied in the system.

5.3.2.2 Local Search Stage

In GRASP, after constructing a solution, the generated solution is further intensified for possible improvements. Particularly for scheduling and routing problems, which simultaneously contain an assignment and a permutation problem, there is a wide range of local search procedures reported in the related literature (Armentano & de Franca Filho, 2007).

In the present study, as mentioned earlier, the aim is to find fast and applicable solutions. Therefore, the neighborhood generations structures, which relatively require lesser effort are given priority here. The used structures are presented in the following.

5.3.2.2.1 Swap. In swap local search, one machine with at least two assigned jobs is randomly selected. Next, two randomly chosen jobs of this machine are switched. This procedure is illustrated in Figure 5.5.a.

5.3.2.2.2 Interchange. In interchange procedure, two machines, each with at least one assigned job, are chosen randomly. Next, two random jobs are chosen from these machines and they are switched. This procedure is illustrated in Figure 5.5.b.

5.3.2.2.3 Insert. In insert local search procedure, one job is randomly chosen from any of the machines and then it is inserted to any position in any machine. This procedure is illustrated in Figure 5.5.c.

It is worth stressing that, eligibility constraints, should be given special attention while performing the interchange and insert local search moves. In the present study, the proposed algorithm considers this circumstance and violating moves are not allowed.

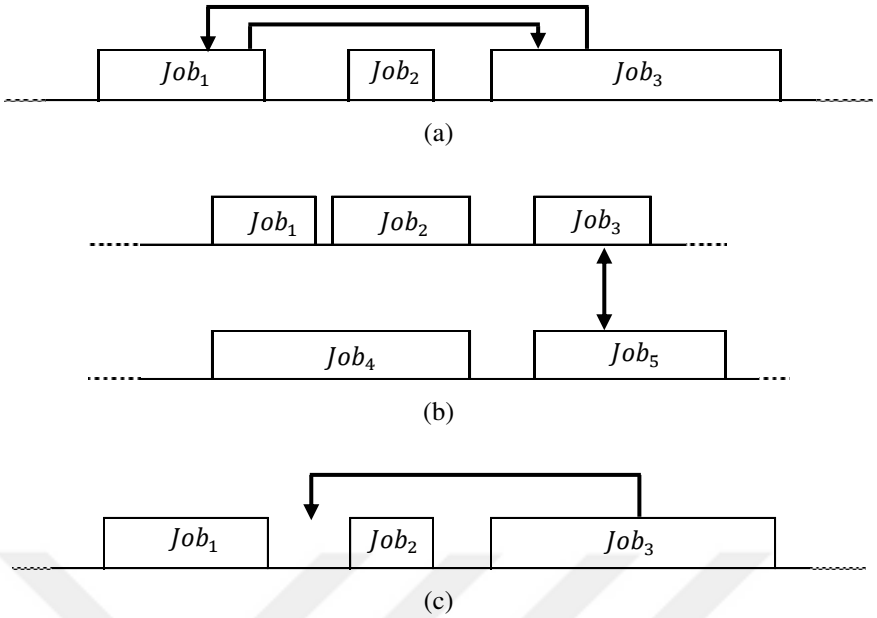


Figure 5.5 a) Swap local search (before swap: 1-2-3, after swap: 3-2-1) b) Interchange local search (before interchange 1-2-3 / 4-5, after interchange: 1-2-5 / 4-3) c) Insert local search (before insert: 1-2-3, after insert: 1-3-2)

According to the preliminary work, it is concluded that there is higher possibility in finding solutions of higher quality, when local search procedures interchange, insert and swap are performed in sequence. Therefore, they are implemented in this sequence in the proposed algorithm.

5.4 Implementation in the Firm

5.4.1 An Application on ERP System of the Firm

In the past period, all schedules of the heat treatment department have been being generated intuitively and based on the experience of the related personnel in production planning department, without using a scientific approach. In other words, what was done in the past was to try to optimize an NP-hard problem under dynamism factors as depicted in Figure 5.6.

The developed algorithm is embedded to the ERP system of the firm as an explicit scheduling algorithm (Figure 5.7). Although the currently in-service ERP system involves some well-known dispatching rules like EDD, SPT and WSPT in its default

settings, these rules have to be modified according to the specific production constraints and processes of any individual firm. Therefore, before using these algorithms, required changes have to be performed in source codes of these rules. The programming language of this system is TROIA, which is currently built on JAVA.

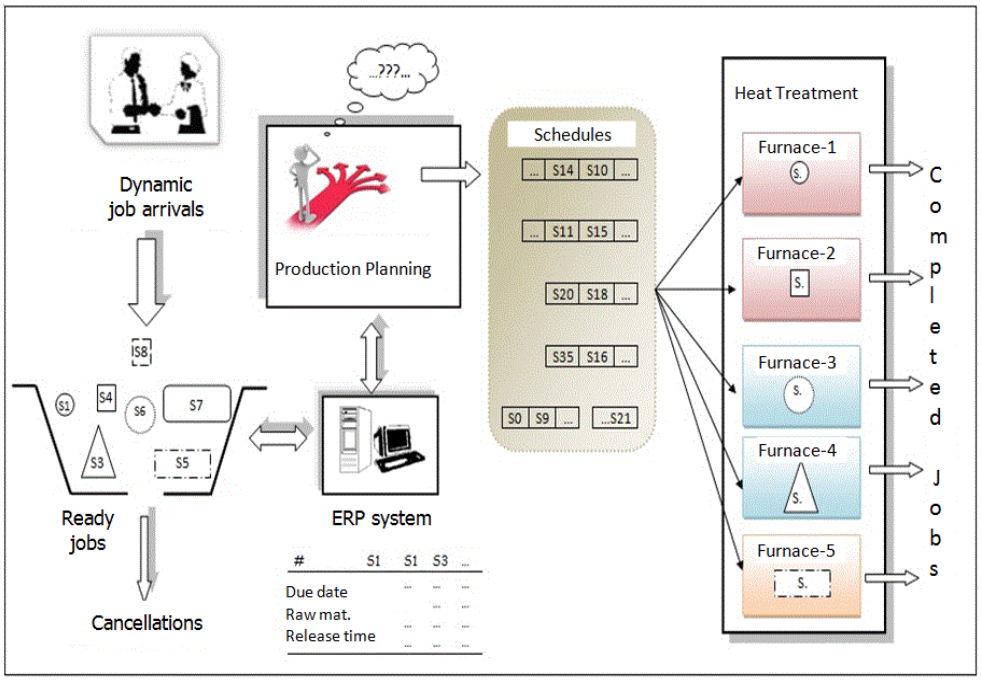
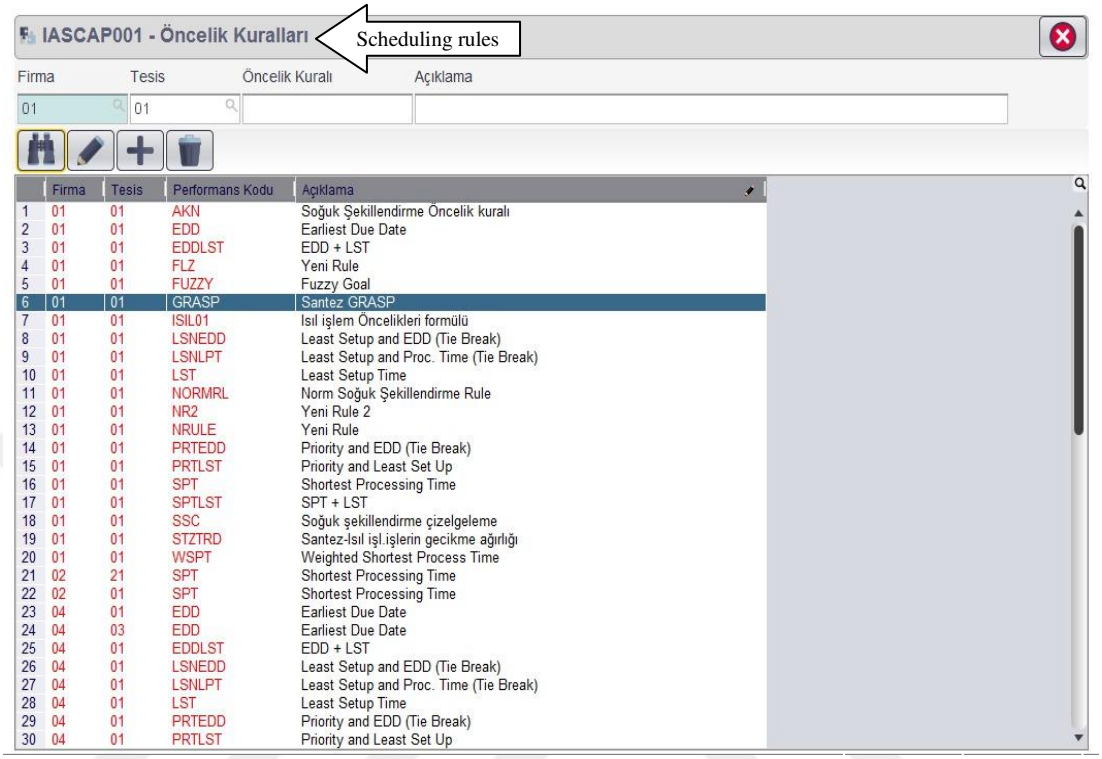


Figure 5.6 A snapshot of the heat treatment department related of whole system

Carried out by the sales department, the first step is accepting orders from customers. Whenever an agreement is fulfilled with a customer, the related order is associated with a unique lot number, involving attributes like due date, lot size, etc. All this data is available in the ERP system of the firm. Let' s assume that a unique lot number A07XZ-LC1QG is defined for a received order. According to this lot number, product, raw material type, supplier of that raw material, setup keys (will be clarified later), due date, eligibility constraints, lot size and process time are also defined to the system. For instance, while the lot number A07XZ-LC1QG is a unique object for the system, A07XZ-LC1QG.*duedate* or A07XZ-LC1QG.*rawmaterial.supplier* are the attributes of this object, which are used by the developed optimization algorithm. Note that eligibility attributes are assigned by the

system automatically. Additionally, if the received order is urgent, it is assigned as the job with red flag.



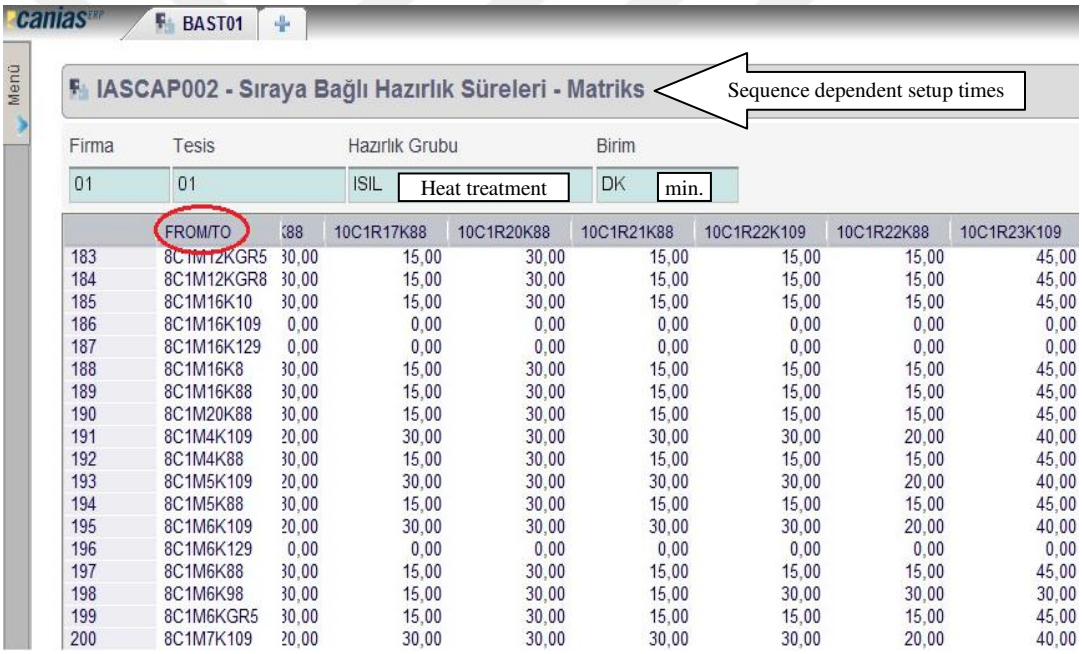
	Firma	Tesis	Performans Kodu	Açıklama
1	01	01	AKN	Soğuk Şekillendirme Öncelik kuralı
2	01	01	EDD	Earliest Due Date
3	01	01	EDDLST	EDD + LST
4	01	01	FLZ	Yeni Rule
5	01	01	FUZZY	Fuzzy Goal
6	01	01	GRASP	Santez GRASP
7	01	01	ISIL01	Isıl işlem Öncelikleri formülü
8	01	01	LSNEDD	Least Setup and EDD (Tie Break)
9	01	01	LSNLPT	Least Setup and Proc. Time (Tie Break)
10	01	01	LST	Least Setup Time
11	01	01	NORMRL	Norm Soğuk Şekillendirme Rule
12	01	01	NR2	Yeni Rule 2
13	01	01	NRULE	Yeni Rule
14	01	01	PRTEDD	Priority and EDD (Tie Break)
15	01	01	PRTLST	Priority and Least Set Up
16	01	01	SPT	Shortest Processing Time
17	01	01	SPTLST	SPT + LST
18	01	01	SSC	Soğuk şekillendirme çizelgeleme
19	01	01	STZTRD	Santez-Isıl işl.işlerin gecikme ağırlığı
20	01	01	WSPT	Weighted Shortest Process Time
21	02	21	SPT	Shortest Processing Time
22	02	01	SPT	Shortest Processing Time
23	04	01	EDD	Earliest Due Date
24	04	03	EDD	Earliest Due Date
25	04	01	EDDLST	EDD + LST
26	04	01	LSNEDD	Least Setup and EDD (Tie Break)
27	04	01	LSNLPT	Least Setup and Proc. Time (Tie Break)
28	04	01	LST	Least Setup Time
29	04	01	PRTEDD	Priority and EDD (Tie Break)
30	04	01	PRTLST	Priority and Least Set Up

Figure 5.7 A view of the program main menu

Defining a setup matrix is a challenging work, particularly if the matrix is asymmetric. Considering the variety of the products and sequence-dependent setup data, it is difficult to determine an error-free data for each of the cells of this matrix.

Although a product has several attributes like length, radius, weight, surface plating, raw material and quality standard, it is concluded by the experts in firm that only three of these attributes have effects on the required setup time between consecutive jobs. These key attributes are radius, raw material type and quality standard. If a combination of all types of attributes including the raw material suppliers is taken into consideration, this combination generates thousands of different products, which will yield to defining a setup matrix with at least 10000x10000 elements. However, note that the most of these products are similar according to the key attributes. Therefore, although the items in this gigantic setup matrix are denoted as different entities, in practice, most of these products are similar based on those three key attributes. As one can see from Figure 5.8, in order to shrink

the size of this matrix, setup keys are defined in the system for once. Each of the unique lot numbers is associated with a single related key. For example, required setup time between jobs with setup keys 8C1M12KGR8 and 10C1R17K88 is 15 minutes (Figure 5.8). By making use of this implicitly clustering method, the size of the setup matrix is reduced to 200x200 elements. According to the specified permutation of the fields in a unique lot number, where the information of raw material, radius and quality standard (key attributes) are embedded, ERP system assigns that lot with the appropriate setup key automatically. For example, consider the setup key 8C1M20K88 that is presented at the 190th row of the FROM/TO column in Figure 5.8. This key means that this product is made up from chrome 8 material, its radius is metric 20 (M20) and its quality standard is K8.8.



	FROM/TO	88	10C1R17K88	10C1R20K88	10C1R21K88	10C1R22K109	10C1R22K88	10C1R23K109
183	8C1M12KGR5	30,00	15,00	30,00	15,00	15,00	15,00	45,00
184	8C1M12KGR8	30,00	15,00	30,00	15,00	15,00	15,00	45,00
185	8C1M16K10	30,00	15,00	30,00	15,00	15,00	15,00	45,00
186	8C1M16K109	0,00	0,00	0,00	0,00	0,00	0,00	0,00
187	8C1M16K129	0,00	0,00	0,00	0,00	0,00	0,00	0,00
188	8C1M16K8	30,00	15,00	30,00	15,00	15,00	15,00	45,00
189	8C1M16K88	30,00	15,00	30,00	15,00	15,00	15,00	45,00
190	8C1M20K88	30,00	15,00	30,00	15,00	15,00	15,00	45,00
191	8C1M4K109	20,00	30,00	30,00	30,00	30,00	20,00	40,00
192	8C1M4K88	30,00	15,00	30,00	15,00	15,00	15,00	45,00
193	8C1M5K109	20,00	30,00	30,00	30,00	30,00	20,00	40,00
194	8C1M5K88	30,00	15,00	30,00	15,00	15,00	15,00	45,00
195	8C1M6K109	20,00	30,00	30,00	30,00	30,00	20,00	40,00
196	8C1M6K129	0,00	0,00	0,00	0,00	0,00	0,00	0,00
197	8C1M6K88	30,00	15,00	30,00	15,00	15,00	15,00	45,00
198	8C1M6K98	30,00	15,00	30,00	15,00	30,00	30,00	30,00
199	8C1M6KGR5	30,00	15,00	30,00	15,00	15,00	15,00	45,00
200	8C1M7K109	20,00	30,00	30,00	30,00	30,00	20,00	40,00

Figure 5.8 A screen shot of the sequence dependent setup matrix

The interface of the developed algorithm is presented in Figure 5.9. Using this interface, a decision maker can define the maximum number of generations the algorithm is run (CIMAXITERATION) as well as the maximum number of generations the local search procedures are performed (CIMAXLOCALSEARCH). The parameters CDCBASECOMPTOLERANCE and CIISRUNCODEOPRFILTER are used for filtering the unnecessary and outdated data.

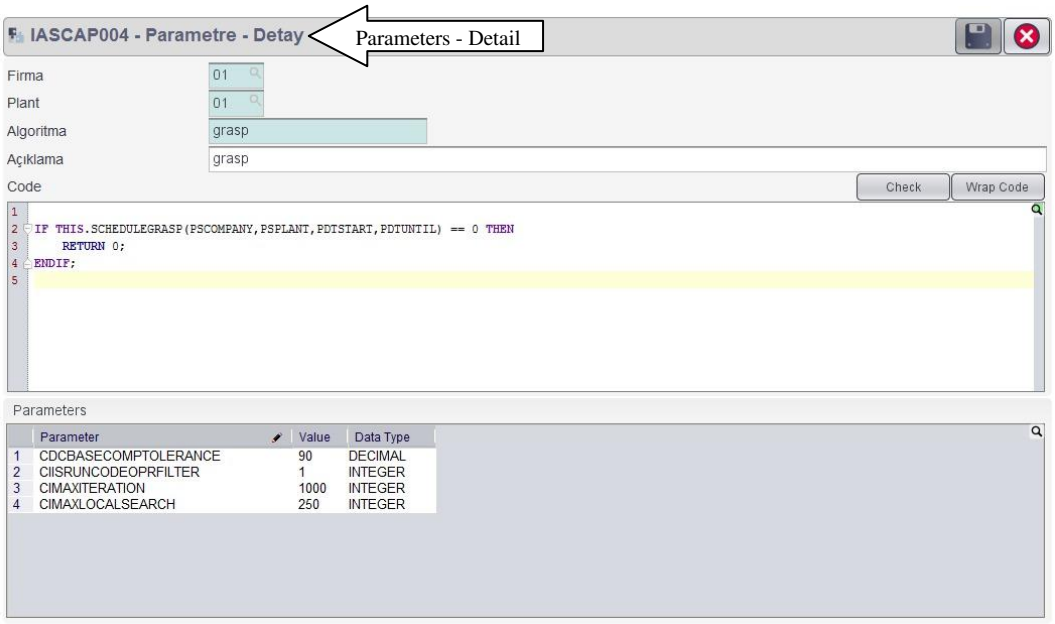


Figure 5.9 A screen shot of the algorithm interface

In the proposed approach, the developed algorithm has three different run modes as periodic, manual and self-directed, where the developed algorithm is triggered periodically due to a predefined interval, or additionally it can be triggered manually or autonomously, respectively. Any authorized decision maker or manager is able to overview the production in heat treatment process as well as they can asses output performances of the furnaces, which are evaluated using the data gathered throughout heat treatment process (Figure 5.10). Furthermore, because the ERP system is also connected to other companies of the same firm, this control is also available from any of those companies.

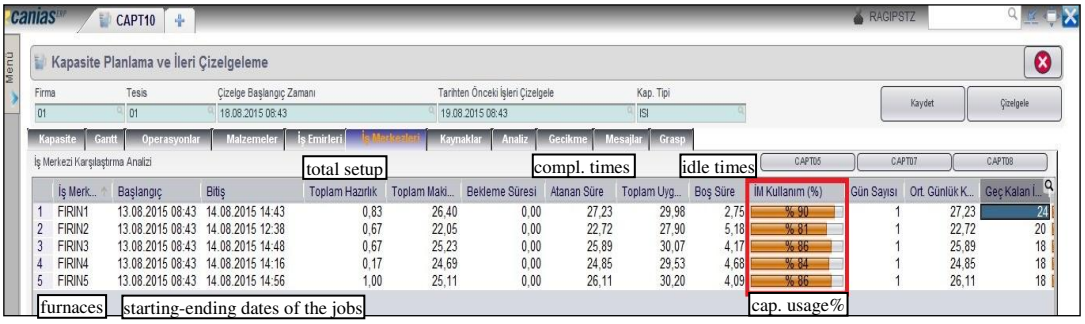


Figure 5.10 An snapshot of online output assessment of an arbitrary schedule

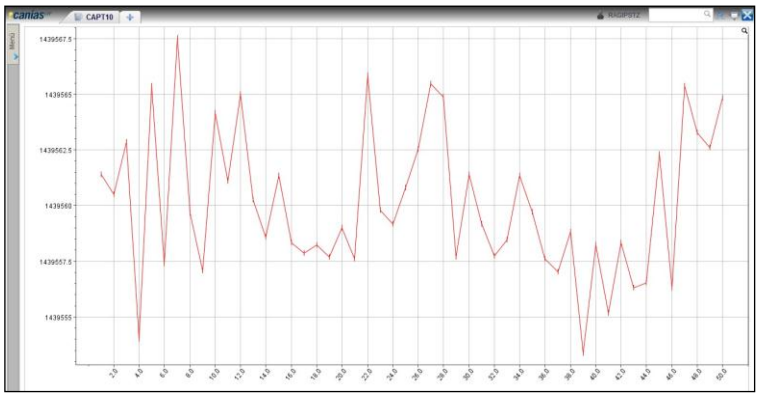
Subsequent to completion of the optimization process of any arbitrary problem environment, obtained Gantt Chart and convergence graphs of the algorithm for this

problem are presented in Figure 5.11 and Figure 5.12, respectively. Although Gantt Chart is more essential for the heat treatment department management, convergence graphs provide some clues about the success of the algorithm. In case of a non-improving graph, the authorized engineers of the heat treatment process are able to re-configure the parameters of the algorithm and seek for different or possibly more qualified schedules for heat treatment furnaces. In this figure, *x*-axis denotes the generation counter while *y*-axis corresponds to objective value. As one can see from Figure 5.11, by clicking on furnace5, the details and sequence of the assigned jobs to this furnace can be seen. The red jobs here represent the tardy jobs. However, it is worth stressing that due dates of the jobs used in the algorithm are updated arbitrarily by the production department in order to prevent the tardiness in worst-case scenarios, which means that most of those jobs are not tardy in practice.

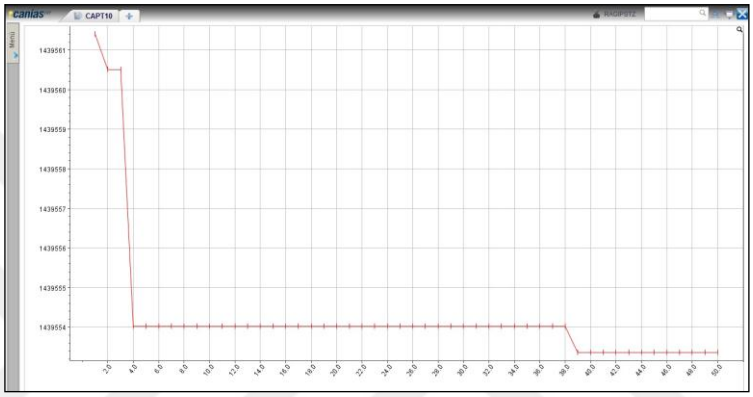
Figure 5.11 An illustrative Gantt Chart for an arbitrary schedule of the furnaces

5.4.2 Experimental Results

In this section, first, effectiveness of the proposed approach is demonstrated through the comparisons with lower bounds, borrowed from the literature. Next, details of the outcomes of the case study by making use of the proposed approach are presented.



(a)



(b)

Figure 5.12 Convergence graph of the developed GRASP algorithm a) plotting the found solution at each generation b) plotting the best-found solution so far

5.4.2.1 Comparison with Lower Bounds

Excluding the maximum number of generations (*genMax*) and the maximum number of local search iterations (*localMax*) employed in GRASP, the proposed algorithm has only three parameters, which affect solution quality. These parameters are α, k_1 and k_2, where k_1 and k_2 are indeed look-ahead parameters of the greedy rule used in construction stage of GRASP. Therefore, first, the appropriate values of α, k_1 and k_2 are attempted to be determined while fixing the parameters *genMax* and *localMax* to arbitrarily sufficient values. According to the preliminary work, it is observed that when k_1 and k_2 are both fixed to 0.1 and α is set to 0.3, better results might be obtained. Therefore, in the proposed approach, the parameters k_1, k_2 and α are fixed to those corresponding values. Finally, fixing these parameters, *genMax* and *localMax* are gradually increased, until no further improvement in solution

quality is achieved. Thus, *genMax* and *localMax* are fixed to 150 and 4500, respectively. The same value of *localMax* is used for each of the local search procedures.

To the best of our knowledge, a similar problem to the handled problem in the present study, is not reported in the related literature. Therefore, in order to demonstrate the effectiveness of the algorithm, the mentioned problem is reduced to a simplified modification, which is denoted by $P \mid s_{jk},\ r_j \mid L_{max}$, for which a benchmark data and a lower bound technique already exist. Benchmarks for this problem are provided by Ovacik & Uzsoy (1995a), Ovacik & Uzsoy (1995b) and a tight lower bound for the same problem is formulated by Ying & Cheng (2010). The mentioned lower bound is given in Equations 5.4-5.6.

$$LB = \max_{i=1,\dots,n}\left\{\min_{j=1,\dots,n} L_{ij}^{LB}\right\} \tag{5.4}$$

where L_{ij}^{LB} denotes the lower bound of the *i*th job when it is scheduled in the *j*th position and it is calculated as follows:

if j=1

$$\begin{aligned} L_{ij}^{LB} &= \max\{C_i - d_i, 0\} \qquad \forall i = 1, \dots, n \\ &= \max\{(r_i + s_{0i} + p_i) - d_i, 0\} \qquad \forall i = 1, \dots, n \end{aligned} \tag{5.5}$$

if j>1

$$\begin{aligned} L_{ij}^{LB} &= \max\left\{C_{ij}^{LB} - d_i, 0\right\} \quad \forall i = 1, \dots, n \text{ and } \forall j \\ &= 2, \dots, n \end{aligned}$$

$$= \max\left\{\left[\max\left\{r_i, R(j), \left[R(1) + S(0) + \sum_{k=2}^{j-1} S(k-1) + \sum_{k=1}^{j-1} P(k)\right]\right\} + S_i(0) + p_i\right] - d_i, 0\right\} \tag{5.6}$$

$$\forall i = 1, \dots, n \text{ and } \forall j = 2, \dots, n$$

In Equations 5.4-5.6, n denotes the number of jobs. In Equation 5.5, s_{0i} represents the initial setup time of the ith job, occurs when that job is scheduled to any idle machine in the first position. $R(k)$ and $P(k)$ $(k = 1, \dots, n)$ correspond to the kth smallest values of the release times and process times for all jobs, respectively. $S(k)$ $(k = 1, \dots, n)$ denotes the kth smallest setup value among all setup times shown by s_{ij} $(i = 1, \dots, n; j = 1, \dots, n; i \neq j)$. Finally $S(0)$ is the smallest value of initial setups s_{0i} $(i = 1, \dots, n)$ and $S_i(0) = \min_{j=1,\dots,n;i\neq j} S_{ji}$ is the smallest setup time of the ith job.

The benchmark data is randomly generated by the method given in details in Ovacik & Uzsoy (1995b), however, in the present study, the same data first generated by Ovacik & Uzsoy (1995a) is used. Each problem of this data set is characterized by the number of jobs n (n={25, 50, 75, 100, 125, 150}) the number of machines m (m={2, 4, 6}) and release time range parameter Γ (Γ={0.2, 0.6, 1.0, 1.4, 1.8}) and 20 instances are available for each of these benchmark problems. Each combination with 20 instances generates 1800 benchmark instances in total.

According to the observations performed in the heat treatment department of the firm, it is concluded that, as a daily average, 40 to 60 jobs are processed. Therefore, in the present study, in order to test the effectiveness of the proposed approach, only the instances with 25, 50 and 75 jobs are used. As a result, in total, 900 instances of this benchmark data are considered in the present study. All tests are conducted on a PC with i7 processor and 16GB ram. The obtained results are presented in the following in Table 5.1.

Table 5.1 Computational results of the proposed algorithm

problem combination	*Avg.*	*Opt.*
(0.2, 25-75, *)	1.834	6
(0.6, 25-75, *)	1.187	26
(1.0, 25-75, *)	1.059	41
(1.4, 25-75, *)	1.053	41
(1.8, 25-75, *)	1.042	62
(*, 25, *)	1.109	79
(*, 50, *)	1.189	60
(*, 75, *)	1.407	37
(*, 25-75, 2)	1.503	37
(*, 25-75, 4)	1.130	71
(*, 25-75, 6)	1.078	59
(*, 25-75, *)	1.236	176

In Table 5.1, encoded by a representation of (Γ, n, m), the column *problem combination* shows the considered instances while evaluating the corresponding performance of the proposed algorithm. For instance, (*, 50, *) means that consider all instances with only 50 jobs while (1.4, 25-75, *) means considering specific instances with Γ=1.4, n={25, 50, 75} and m={2, 4, 6}. The column *Avg*. is evaluated as the average of the best solutions found by the algorithm, to the lower bound values over all instances of the corresponding problem combination. Finally, the column *Opt*. is the number of instances, for which the best solution found by the algorithm is equal to the lower bound of that instance.

If the first 5 rows of the *Avg*. column are considered, it can be seen that near optimal results are obtained for each the parameter Γ except for the Γ=0.2 which generates relatively more challenging instances of this class. According to the rows 9-11 of the same column, one can see that one of the most successful results of this data set for (*, 25-75, 6), which severely resembles to the problem handled in case study, is obtained. Finally, considering the final row of the same column, it can be concluded that the proposed algorithm can possibly either derive optimal solutions or near optimal solutions with a maximum average of 23% gap to the theoretical lower bound presented in Equations 5.4-5.6.

5.4.2.2 Results of the Case Study

Expenditure items of operating a heat treatment furnace in the firm, where the project is conducted, are presented in Table 5.2. Thus using these values, it can be calculated that a furnace is operated with an approximate cost of 1.39 €/min.

Table 5.2 The cost of operating a heat treatment furnace

Depreciation expense	120 € / 1000 min.
Indirect labor cost	170 € / 1000 min.
Direct labor cost	260 € / 1000 min.
Operating cost	160 € / 1000 min.
General production expenses	680 € / 1000 min.

According to the pilot study carried out in the firm, it is clearly observed that by making use of the developed algorithm, total idle time of the furnaces, either because of the setups, due to previously unorganized schedules or because of other

operational inappropriate preferences, is decreased approximately 3080 minutes in a monthly period. This results in as 36960 minutes decrement annually in total idle time of the furnaces. As a result, assuming that additional production is not performed in this gained additional period of 36960 minutes, the decrement in total idle time of the furnaces yields to 51374.40 € savings annually.

On the other hand, assuming that the production capacity of heat treatment department is 1.26 tons/hr., utilizing this additional time (36960 min.), heat treatment department can now produce up to an additional 776.16 tons of fasteners annually. Average sales price of the fasteners produced in the firm is approximately 1825 €/ton. Assuming that the required demand will be available and any unforeseen bottlenecks do not occur, it can be calculated that if the heat treatment department is run in this additional period, annually an additional 1416492 € is expected. To sum up, by making use of the proposed algorithm, the annual gain is expected to be between 51374.40 € and 1416492 €.

5.5 Chapter Conclusion

In this chapter, a case study of online and dynamic scheduling of parallel heat treatment furnaces in a manufacturing company is presented. The focused problem is a challenging extension of the well-known parallel machine scheduling problem, including sequence-dependent setup times, release times, due dates, weights, eligibility constraints, machine breakdowns/maintenance and real-life dynamism factors like arrivals of rush orders, cancellations of already handled jobs or changes in any problem related parameters like due date or lot sizes.

A constructive and multi-start strategy is proposed to solve the mentioned problem. Thus, while guaranteeing feasible and applicable schedules in short time period, which is a crucial in real-life applications, additionally challenges of combinatorial dynamic optimization problems are handled easily.

The proposed algorithm is embedded to the ERP system of the firm as a module. The developed module has three different run modes such as periodic, manual and self-directed, where the developed algorithm is triggered periodically due to a

predefined time interval, or additionally it can be triggered manually or autonomously, respectively.

In order to demonstrate the effectiveness of the proposed approach, the results of the algorithm are compared with a tight lower bound technique, already formulated in the related literature. It is concluded that efficient results might be obtained by the proposed approach.

Finally, the expected outputs of the firm are assessed in the present chapter. With the use developed and integrated algorithm that employs a multi-start and constructive strategy along with optimization techniques, annually, an additional income between 51374.40 € and 1416492 € is expected.

CHAPTER SIX
MINIMIZING TOOL SWITCHING AND INDEXING TIMES WITH TOOL DUPLICATIONS IN AUTOMATIC MACHINES

This chapter presents another industrial application of DOPs, particularly in flexible manufacturing systems, where generally medium-sized volumes of a variety of part types are processed by numerically controlled machines. In this manufacturing system, minimization of non-machining time is a crucial issue for effective and profitable utilization of automatic machining centers. Additionally, a fast response to possible changes in production is vital in order to attain flexibility. In this context, first, an effective strategy is developed to generate efficient policies to be used in those machining centers. Next, the mentioned problem is handled as a dynamically changing problem and the proposed method is extended to be able to handle dynamic events. Extensive experimental study is provided in this chapter.

6.1 Chapter Introduction

"*A flexible manufacturing system (FMS) is an integrated, computer controlled complex of automated material handling devices and numerically controlled (NC) machine tools that can simultaneously process medium-sized volumes of a variety of part types*" (Stecke, 1983). In such systems, minimizing non-machining times on automatic machining centers is a crucial issue for operation managers in order to operate these expensive machine tools profitably and to shorten the return on investment time. Automatic tool changer (ATC) is one of the main components of automatic machining centers. Optimizing its operations can considerably reduce non-machining time. There are actually two major operations on ATCs, which can be reduced by optimization procedures. These operations are known as *ATC indexing problem* and *tool switching problem* (ToSP) on a single machine. These two challenging problems are generally addressed separately in the related literature. However, rather than solving these operations as two different problems, handling them simultaneously is vital in minimizing total non-machining times in machining centers.

Makespan on a single machine is actually the summation of machining (cutting operations) and non-machining (ATC indexing times and sequence-dependent setup times due to tool switches) times. Cutting times of the parts, which form up the machining times, can be considered constant (they can simply be ignored in optimization procedure) because the problem environment is a single machine case. Non-machining times, on the other hand, directly affect the makespan because they occupy rest of the production time. Therefore, part sequence and cutting tool allocation policy on the ATCs clearly have effects on makespan. Moreover, ATC indexing and ToSP problems should be handled simultaneously because there might be some occasions that an intentional worse indexing policy might result in better setup times or vice versa.

ToSP (Crama, Kolen, Oerlemans & Spieksma, 1994; Tang & Denardo, 1988) commonly arises in FMSs (Bordoloi, Cooper & Matsuo, 1999; Browne, Dubois, Rathmill, Sethi & Stecke, 1984; Stecke, 1983), where different sets of cutting tools are required for each of the parts. In this problem, because the capacity of ATC is limited and the total number of the required tools to process all parts might exceed this capacity, in order to meet the requirements of upcoming parts, while some tools are replaced (switched) with other ones, some of them might be kept on their current positions. ToSP is indeed a combination of two interrelated problems commonly referred as sequencing and tooling. The aim here is to find the best possible sequence of parts so that the minimum number of tool switches is obtained. For a given sequence, there might be several tooling policies, however, an algorithm referred as "*Keep Tool Needed Soonest*" (Tang & Denardo, 1988) can solve this stage optimally, where in tooling, the decision either to keep or to switch a tool is made. ToSP is proven to be NP-complete (Crama, Moonen, Spieksma & Talloen, 2007; Djellab, Djellab & Gourgand, 2000).

There is a variety of solution approaches for ToSP in the related literature. In one of earliest studies, Hertz, Laporte, Mittaz & Stecke (1998) propose several heuristics. Keung, Ip & Lee (2001) employ a GA for this problem. Laporte, Salazar-Gonzalez & Semet (2004) compare the performances of branch-and-cut and branch-and-bound algorithms in ToSP. In a following study, tool size and magazine capacity constraints

are considered in van Hop (2005). Salonen, Raduly-Baka & Nevalainen (2006) propose a multi-start search strategy that tries to avoid from local optima traps. They test their algorithms on randomly generated benchmarking problems and on real-life data. In some other studies, (Akturk & Ozkan, 2001; Avcı & Akturk, 1996; Sinriech, Rubinovitz, Milo & Nakbily, 2001) a similar problem is solved as a sub-problem of scheduling in FMSs. In another related study, Matzliach (1998) solve ToSP in a dynamic environment and proposes three heuristics with several assumptions. Unfortunately, in the majority of the ToSP related studies, ATC indexing time, which is one of the key factors in minimizing non-machining times, is ignored.

ATC indexing problem, in which the aim is to find the optimum allocation of the required cutting tools on the stations (positions, pockets, slots, etc.) on a turret magazine of a CNC machine, is one of the most widely encountered challenging problems in real-life machining industry. As already noted by Dereli & Filiz (2000), determination of the best possible positions of cutting tools on tool magazines (ATC, turret magazine, etc.), is crucial for reducing non-machining times.

Each part produced in a manufacturing system might require different set of cutting tool types in various sequences through their machining operations. The sequence of the corresponding required cutting tool type forms up the process plan of each of these individual parts. According to these process plans, switching from one cutting tool to another one rotates the turret magazine from one position to the other one. The time elapsed during this rotation is known as indexing time in the related literature. An illustration of a turret magazine, stations on magazine and indexing time is presented in Figure 6.1.

Utilizing the advances of today's technology, although machining plants have equipped their machines with more enhanced equipments, the mentioned indexing time still can be reduced by applying effective tool allocation policies in order to decrease non-machining time (Dereli, Baykasoğlu, Gindy & Filiz, 1988; Dereli & Filiz, 2000), which directly affects the total production time and cost. Reducing indexing time even becomes more important for the machines that cannot provide a rapid tool indexing capability. Additionally, it is obvious that one remarkable

approach to reduce ATC indexing time is to place available copies (duplicates) of cutting tools on the possible empty positions of turret magazine. Although, some tools for some particular tasks might be too expensive to afford, duplications of some other appropriate cutting tools might clearly reduce ATC indexing time. As stated by Gray, Seidmann & Stecke (1993), "*An important, unanswered research problem involves determining the optimal number of copies of each tool type to load into a magazine*". However, most of the studies ignore possible tool duplications surprisingly and the related literature is quite lacking in this respect.

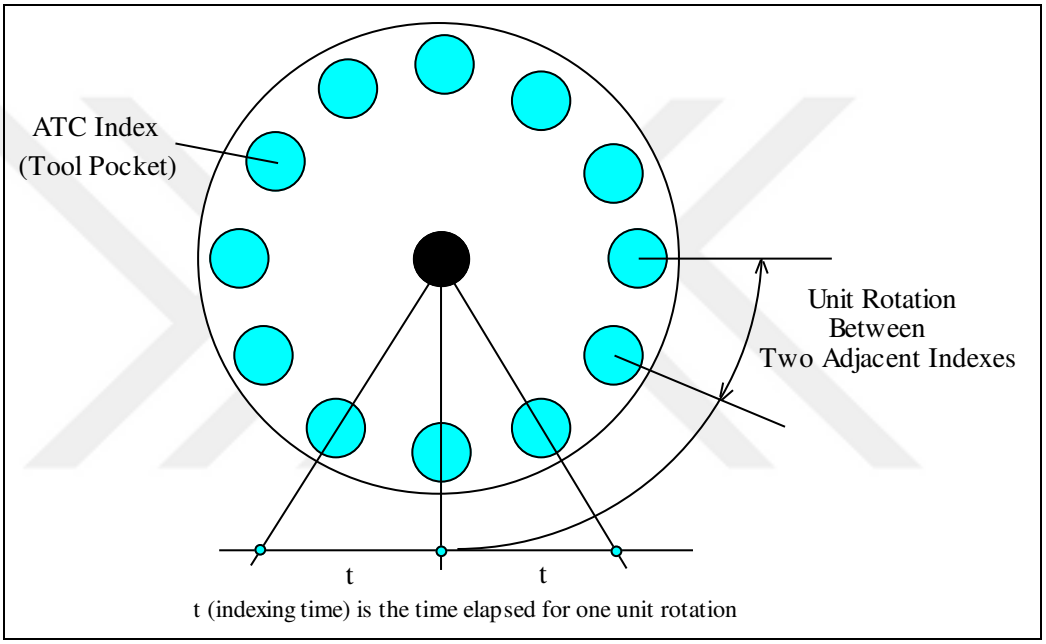


Figure 6.1 Tool indexing on a turret magazine (Dereli & Filiz, 2000)

Dereli & Filiz (2000) and Dereli et al. (1998) report one of the first attempts in addressing the problem mentioned above. The authors propose a GA as a solution approach. However, their algorithm ignores tool duplications. In their studies, the problem is modeled as an assignment problem with a complex non-linear objective function, which calculates total indexing time. Wilson (1987) reports another similar work, where a different objective function is used. Levitin & Rubinovitz (1995) solve ATC indexing problem, by using an extended nearest neighborhood procedure, with different weight parameters. They test their algorithms on a large set of randomly generated test problems. In another study, Turkcan, Akturk & Storer (2007) propose heuristics that are used in a GA to solve loading, scheduling, and tool

management problems simultaneously in FMSs with parallel CNC machines. In that study, two objectives are considered as minimization of manufacturing cost and total weighted tardiness. The authors employ a tool-life-based approach to define the tool placement policy. Baykasoğlu & Dereli (2004) introduce a strategy that considers tool duplications. Their approach simultaneously decides the number of existing copies and allocations of each of the cutting tools on the turret magazine. The authors employ a simulated annealing (SA) algorithm to solve ATC indexing problem. In the following study, Dereli & Baykasoğlu (2005) further improve the proposed planning system of Dereli & Filiz (1999), that can produce optimal process plans with several novelties as well as consideration of cutting tool duplications. Finally, in one of the latest studies in this research field, Baykasoğlu & Ozsoydan (2016) extend the study of Baykasoğlu & Dereli (2004) by making use of a shortest path algorithm in order to evaluate the indexing time of a turret magazine for a corresponding tool allocation policy. However, the mentioned problem is solved for a single part, as a result possible tool switches between parts in a multi part machining scenario is not considered in that study.

The present chapter contributes to the previous studies by presenting a solution approach for simultaneously solving ToSP and ATC indexing problems with tool duplications. By handling ATC indexing with tool duplications and ToSP simultaneously, more realistic conditions and well-defined non-machining times are revealed for these problems. By making use of the proposed strategy, ToSP, which is a stand-alone NP-complete class problem, is implicitly solved without further effort. Moreover, as to be clarified later in details, the mentioned problem has a unique feature that is; objective function evaluation for this problem requires a shortest path algorithm to find the exact ATC indexing time for a corresponding policy. A SA algorithm is employed to solve this problem on a set of generated benchmarks. Due to the unavailability of a similar work in the literature, obtained results are compared with the lower bounds, proposed for those generated benchmarks.

The rest of the this chapter is organized as follows: Section 6.2 is devoted to problem definition and algorithm related procedures, while the details of the proposed solution approach to solve the mentioned problem are presented in Section

6.3. Experimental study along with concluding remarks are given in Section 6.4, Section 6.5 and Section 6.6, respectively.

6.2 Problem Definition and Algorithm Related Mechanisms

The present problem is at the intersection of ATC indexing problem and ToSP as it is shown in Figure 6.2 by the shaded area. The ultimate goal here is to minimize the makespan by reducing the total non-machining time, which is the summation of ATC indexing times of each part and tool switching time between parts, according to a schedule.

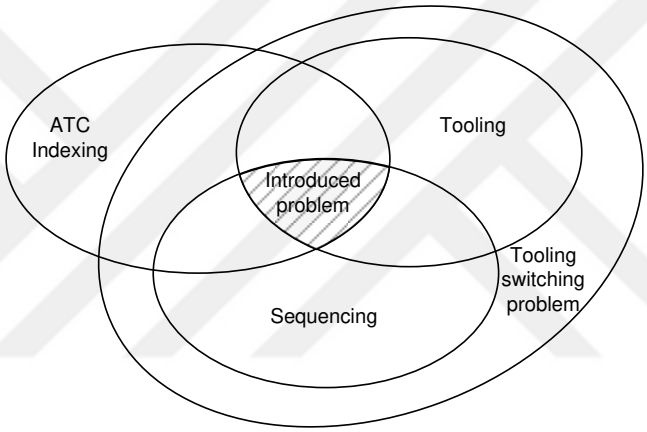


Figure 6.2 A relationship diagram between the mentioned problems

For a given process plan, ATC indexing time of a part is calculated by using a shortest path algorithm which was previously proposed by the authors (Baykasoğlu & Ozsoydan, 2016). After ATC indexing time of each part is calculated, for a given sequence, the total number of tool switches between parts, which is the other term of non-machining times, is calculated. The hierarchy of the mentioned problem is depicted in Figure 6.3.

Finally, an overall illustration of the problem is presented in Figure 6.4. As one can see from this figure, a party is comprised of the same types of parts. For example, the first party processed in the machine is comprised of only PART1 type

of parts and the lot size of this party is equal to three. Similarly, the lot size of the last party with PART4 equals to two. Because the machining operations of a party cannot be interrupted by a different part type, (preemption is not allowed) for simplicity, each party can occasionally be considered as individual parts with predefined lot sizes and this notion is adopted throughout this thesis.

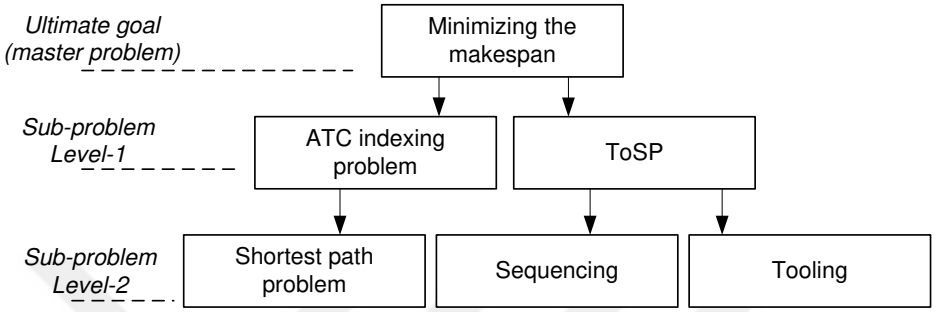


Figure 6.3 Hierarchy of solving the introduced problem

For a given process plan, an employee loads the required cutting tools to the turret magazine. Subsequent to the completion of the machining process of the current part, the same employee replaces the required cutting tools according to the next part in the production queue. The sum of each removing and loading actions on the turret magazine finally gives the occurred setup between parts. It is clear that any tool switching is not incurred between the components of the same party. The calculation procedure is later clarified in the objective function evaluation section, in details.

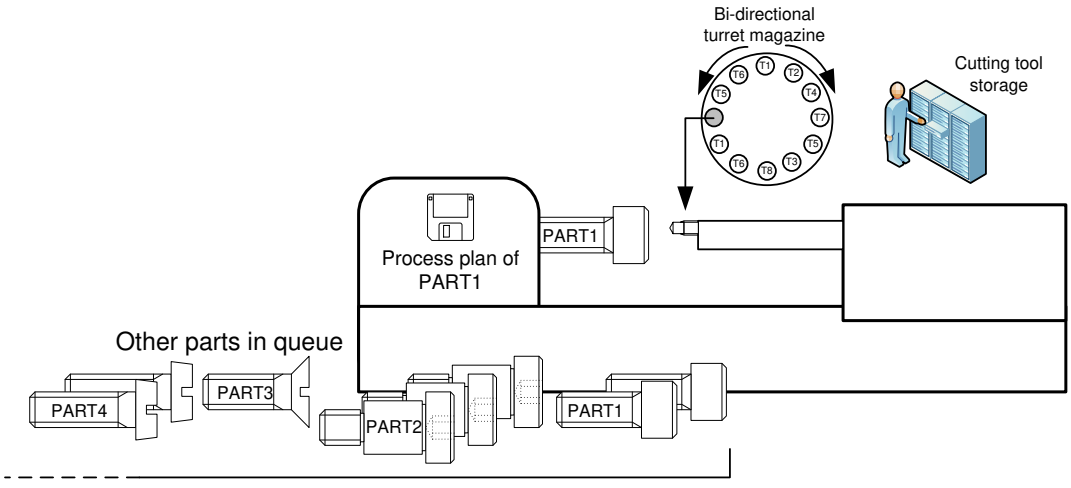


Figure 6.4 An illustration of the problem environment

6.2.1 Encoding-Decoding

The solution representation of Baykasoğlu & Dereli (2004) is adopted here. One notable advantage of this technique is that it simultaneously decides which tools to be duplicated, the number of duplications, and the allocation of all tools on the turret magazine. However, in order to encode a part sequence, this method is further extended by appending a single row tuple. This technique is explained on an example, presented in the following.

Let's assume that there are 4 parts in the problem environment with lot sizes 10, 10, 5 and 25, respectively. Again, let's assume that PART1 has 5 operations and it requires tool-3, tool-4, tool-3, tool-3 and tool-1 in sequence. Similarly, PART2 has 5 operations and it requires tool-1, tool-3, tool-3, tool-2 and tool-1 in sequence. PART3 has 6 operations and it requires tool-4, tool-1, tool-1, tool-2, tool-4 and tool-1 in sequence. Finally, PART4 has 3 operations and it requires tool-3, tool-4 and tool-2 in sequence. For each tool type (tools 1-2-3-4), there exist 3, 2, 2 and 4 copies, respectively in the workshop and there are 8 stations on the turret magazine. According to the given schedule, PART3, PART1, PART4 and PART2 are going to be processed in sequence.

In this technique, the length of a solution representation is dependent to the number of the required tool copies for the corresponding part and to the number of stations on the turret magazine. Therefore, the required length of the solution representations for individual parts might vary so they are kept separately. The length of each of these representations is determined by $max\{\sum_{i=1}^{numberoftools} copiesOfTheRequiredTool_i, numberOfATCindexPositions\}$, i.e. $max\{9,8\}=9$ for PART1, $max\{7,8\}=8$ for PART2, $max\{9,8\}=9$ for PART3 and $max\{8,8\}=8$ for PART4. Each of these representations has two rows, where the second row includes the cutting tool types and the first row includes binary variables for indicating whether the corresponding cutting tool is assigned on the turret magazine or not. After generating matrices with the required dimensions for each part, initially, the first rows are filled by 1s or 0s with an exception. If the number of

stations on the turret magazine is greater than or equal to the total number of tool copies required in the process plan of a part, all stations on the turret magazine are candidate for tool allocation. Thus, each of the bits in the first row are filled with 1s (Figure 6.5.b, d), otherwise, the number of the 1s in the first row is determined by $min\{\sum_{i=1}^{numberoftools} copiesOfTheRequiredTool_i, numberOfATCindexPositions\}$ (Figure 6.5.a, c). The required number of 1s is assigned randomly to the first row. The rest of the unfilled bits in the first row are filled by 0s. Next, at least one of the required tools to complete the operations of the corresponding part is randomly assigned to the second row with 1s above and they are highlighted as the fixed positions, which is an important issue for neighborhood generation. The rest of the second row is randomly filled by the remaining cutting tools. Finally an additional tuple, where the sequence of the parts is kept, is randomly defined (Figure 6.5.e). Now tools can be allocated to the turret magazine only following the 1s in the first row with a simple procedure. For example for PART1, tool-4, tool-3, tool-4, tool-1, tool-3, tool-1, tool-4 and tool-1 are assigned to the *1*st, *2*nd, *3*rd, *4*th, *5*th, *6*th,*7*th and *8*th stations of the turret magazine. Some arbitrary representations for each of the parts in this example and corresponding allocations on the turret magazine are presented in Figure 6.5 and Figure 6.6, respectively. Several more examples are also provided in Baykasoğlu & Dereli (2004) and Baykasoğlu & Ozsoydan (2016).

6.2.2 Neighborhood Generation

The swap strategy, where two randomly chosen bits are exchanged, is employed here as the neighborhood generation technique. It is applied to the first and the second rows in sequence, unless a fixed position is selected. If any of these random positions is one of the fixed positions, all corresponding columns (including the first and the second rows) are swapped rather than swapping only those positions. In this technique, the first row is swapped first. The procedure is defined as follows:

Step-1: Select four random bits from the solution representation of the corresponding part, two from the first row and two from the second row, respectively.

Step-2: If the selected locations are not fixed positions, only the selected elements in the corresponding row are swapped. If a fixed position is selected in any row, then swap the corresponding columns.

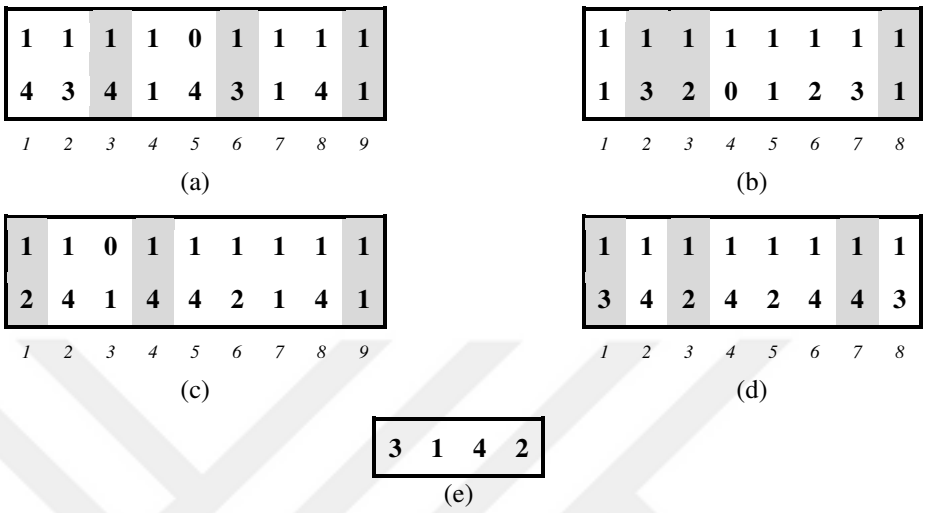


Figure 6.5 Solution representations for a) PART1, b) PART2, c) PART3, d) PART4, e) part sequence

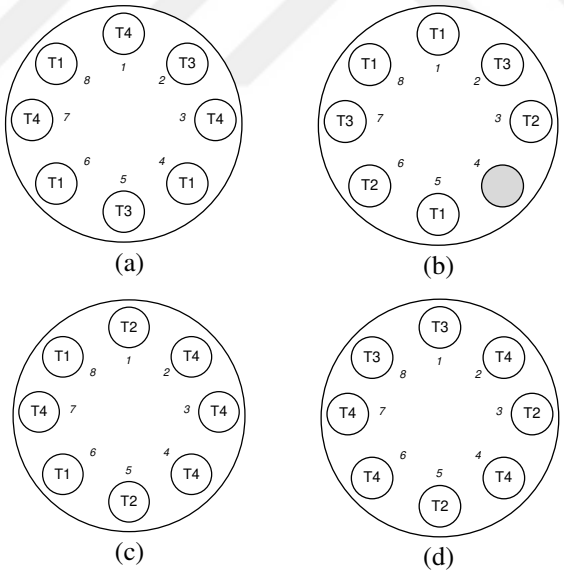


Figure 6.6 Allocation of the cutting tools on the turret magazine for a) PART1, b) PART2, c) PART3 and d) PART4.

Finally, any two random bits in the tuple, representing the job sequence are swapped. All swapping moves are performed on the same neighborhood generation. An illustrative example is presented in Figure 6.7. As one can see from this figure,

randomly selected bits of each representation for the corresponding parts are shaded. Swap procedure, including fixed positions are illustrated in Figure 6.7.a-b, whereas the swaps without fixed positions are presented in Figure 6.7.c-d.

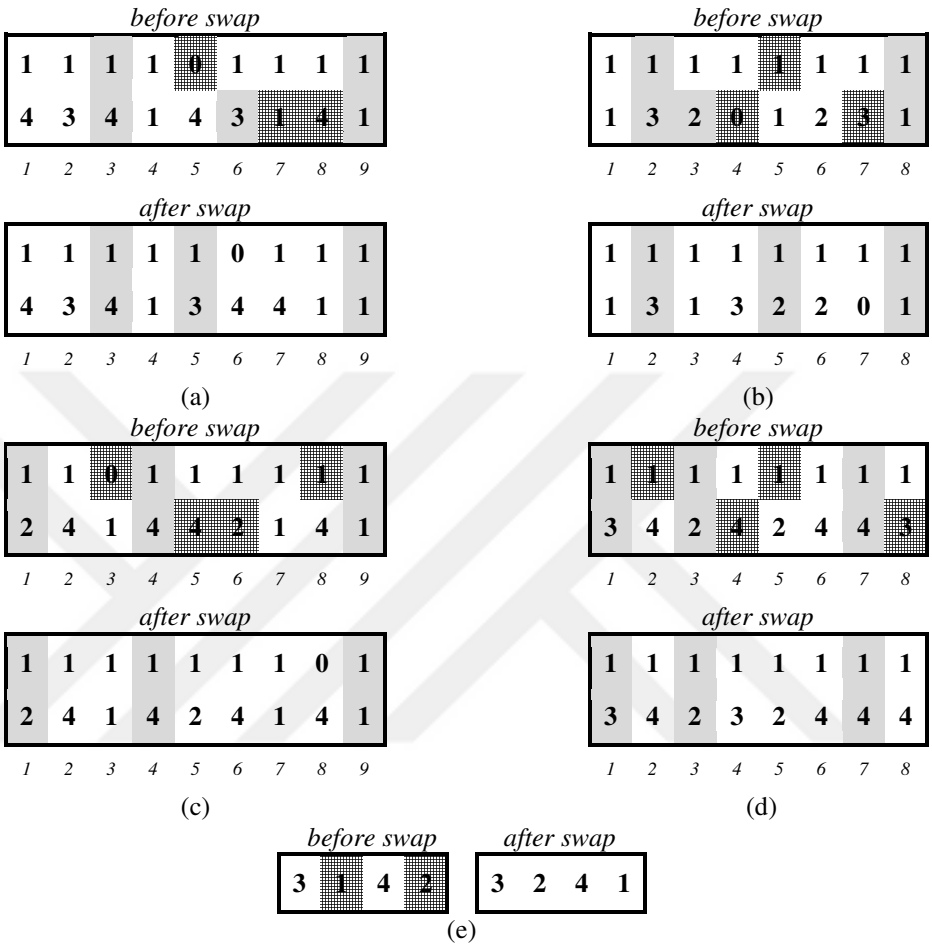


Figure 6.7 New solution representations for a) PART1, b) PART2, c) PART3, d) PART4, e) part sequence

In this procedure, fixed position marking is used to meet the minimum requirements for cutting processes of the parts. In other words, according to the process plan of the corresponding part, at least one copy of each required tool types should be allocated on the turret magazine. Therefore, exactly one copy of each of the required cutting tools, assigned on the turret, is defined as fixed position. However, the positions of these tools on the turret should be able to be changed, which is already allowed by this procedure. Therefore, this mark must not be lost until the end of the optimization process and it is carried to its new positions as in Figure 6.7.a-b. To sum up, this swapping procedure of Baykasoğlu & Dereli (2004),

which is adopted here by further extending it with part sequence tuple, allows both changes of positions of the already allocated cutting tools on the turret magazine (slot changes of the assigned copies on the magazine) and the changes between the assigned and unassigned cutting tools. Additionally it changes the part sequence. All swapping moves including the swap in the part sequence are performed simultaneously, which means that a neighborhood of a solution corresponds to a different job sequence and different tool allocation policies for each of the parts. New tool allocations on the turret magazine for the corresponding parts are depicted in Figure 6.8.

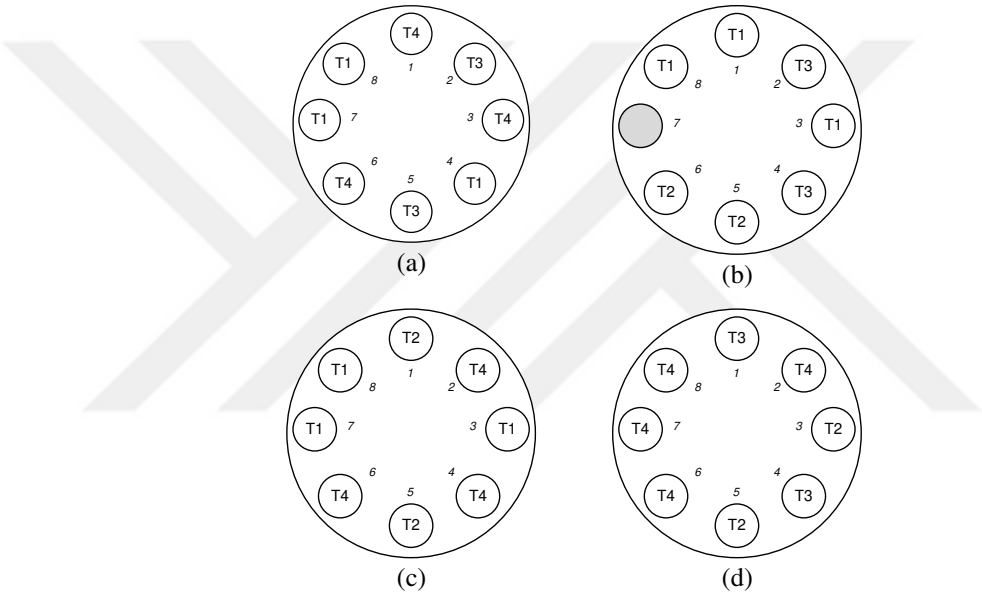


Figure 6.8 New allocations of the cutting tools on the turret magazine for a) PART1, b) PART2, c) PART3 and d) PART4

6.2.3 Objective Function Evaluation

First, non-machining time arising from ATC indexing is calculated in the mentioned problem. The following ATC indexing related assumptions stated in Baykasoğlu & Dereli (2004), are adopted here as well.

- Each cutting tool can be placed in any index position of the ATC and each occupies only one index position.
- The weight balancing of the ATC is ignored.

- Tool interchanging during cutting operations is not allowed.
- A tool is removed from an index position of the ATC by a robot arm and it is returned to the same index position subsequent to cutting operation (Figure 6.4).
- Tool-life of each cutting tool is long enough to complete the planning horizon.

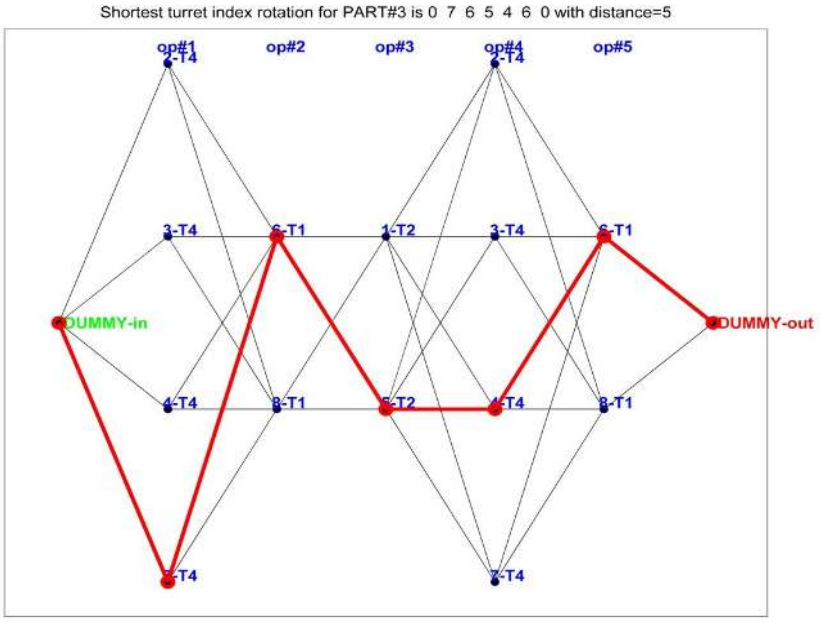
Under these assumptions, first, non-machining time arising from ATC indexing is calculated. For this purpose, a method, where a shortest path algorithm is run on the generated networks for each of the corresponding parts, is used (Baykasoğlu & Ozsoydan, 2016). This technique is exemplified in the following by using the example, which is already presented in Section 6.2.1.

According to the given schedule, first PART3 is processed. PART3 has 6 operations and it requires tool-4, tool-1, tool-1, tool-2, tool-4 and tool-1 in sequence, however, turret is not rotated between the *2*nd and the *3*rd operations because these operations require the same tool type (tool-1). As a result, cutting operations of PART3 is reduced to 5 stages (Figure 6.9.a). The first cutting stage (op#1) requires tool-4 and according to the tool allocation policy of PART3, tool-4 is allocated on the *2*nd, *3*rd, *4*th and *7*th stations on the turret (Figure 6.6.c). Therefore, turret must begin to cutting operations from one of these stations. The next required tool is tool-1 (op#2). According to the allocation policy of Figure 6.6.c again, this tool is allocated on the *6*th and *8*th stations on the magazine. Turret must rotate to one of these stations from the previously chosen station of the previous cutting operation. The next tool (op#3), tool-2 is located on the *1*st and *5*th locations (Figure 6.6.c). Turret rotates again to one of these locations. Next stage (op#4) requires tool-4, which is already located on the *2*nd, *3*rd, *4*th and *7*th stations. Finally, subsequent to operation of tool-1, which is the last stage (op#5), cutting operations of PART3 is terminated.

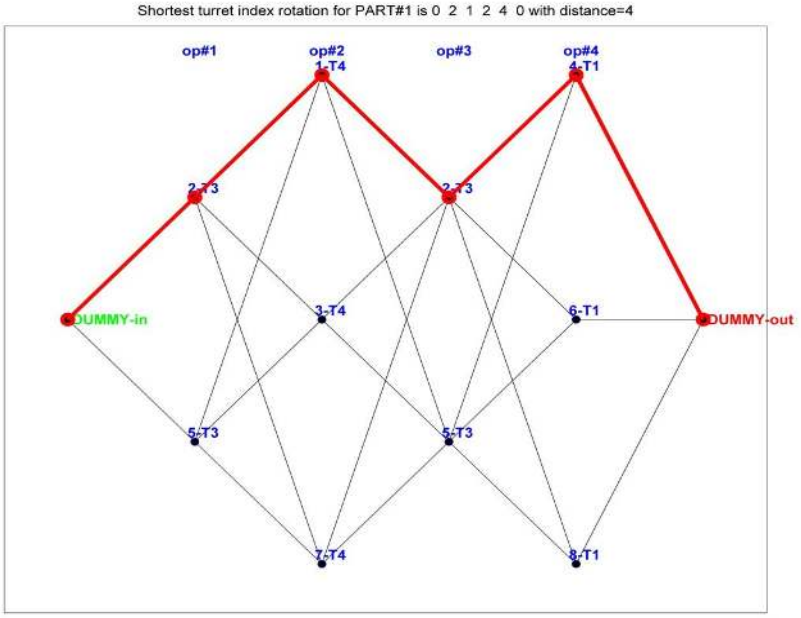
The decisions on the turret rotations, which give the total ATC indexing time, are critical here. Although there are numerous rotation alternatives for a given allocation policy, only one of them gives the minimum number of rotations (assuming that alternative solutions do not exist). For example, at the beginning of the operations of PART3, it is clear that turret can also be rotated from the *2*nd position to the *6*th position while progressing from op#1 to op#2. It means that the *2*nd position, which

is one of the slots occupied by tool-1, is chosen for op#1. However, this rotation yields to 4 unit rotations. Similarly, numerous combinations can be generated in order to process PART3. However, only one of these rotation alternatives gives the minimum total number of rotations. The aim here is to find this minimum route, which gives the exact ATC indexing time of the corresponding part, for a given tool allocation policy. Otherwise, the objective value would be ill-defined. This is assured by implementing a shortest path algorithm to the network obtained by appending the DUMMY-in and DUMMY-out nodes as represented in Figure 6.9. Finally, Dijkstra's algorithm (Dijkstra, 1959) between these dummy nodes, connected with 0-cost edges, is run and the minimum number of rotations for the corresponding tool allocation policy is obtained as highlighted in Figure 6.9. However, it is worth stressing that the optimum tool allocation policy might still not be found. Only the objective value of a corresponding policy is calculated by this algorithm, which is one of the interesting points of the introduced problem. As a result, turret is started from position 7, then it is rotated to the *6*th position in the counter-clock-wise direction (1 unit rotation), to the *5*th position in the counter-clock-wise direction (1 unit rotation), to the *4*th position in the counter-clock-wise direction (1 unit rotation) and finally to the *6*th position in the clock-wise direction (2 unit rotations). In total, 5 rotations is required to process PART3 using the tool allocation policy of Figure 6.6.c. Although indexing time can be different for CNC machines, in the present study it is assumed that each unit rotation takes 1 second. Therefore, for cutting process of PART3, 5 seconds of non-machining time during ATC indexing is required. Multiplying it with the lot size of PART3, 25 seconds ATC indexing time is obtained to complete this party. By evaluating rest of the parts in production queue, in total, it is found that 145 seconds for ATC indexing is required to complete the production.

Evaluation of the sequence-dependent setup times (tool switches) between parts is relatively easier than the above procedure. However, for this problem, there exist some assumptions (Crama et al., 1994) again. These assumptions are given in the following.

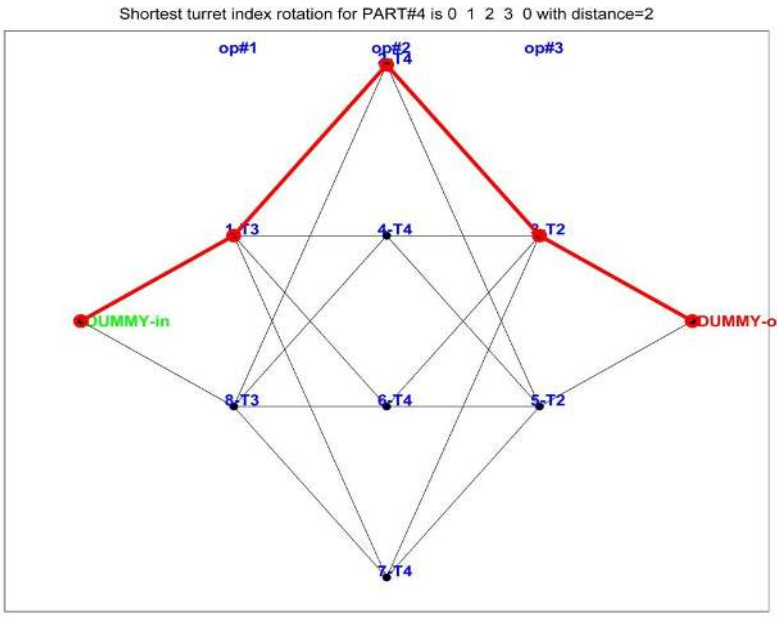


(a)

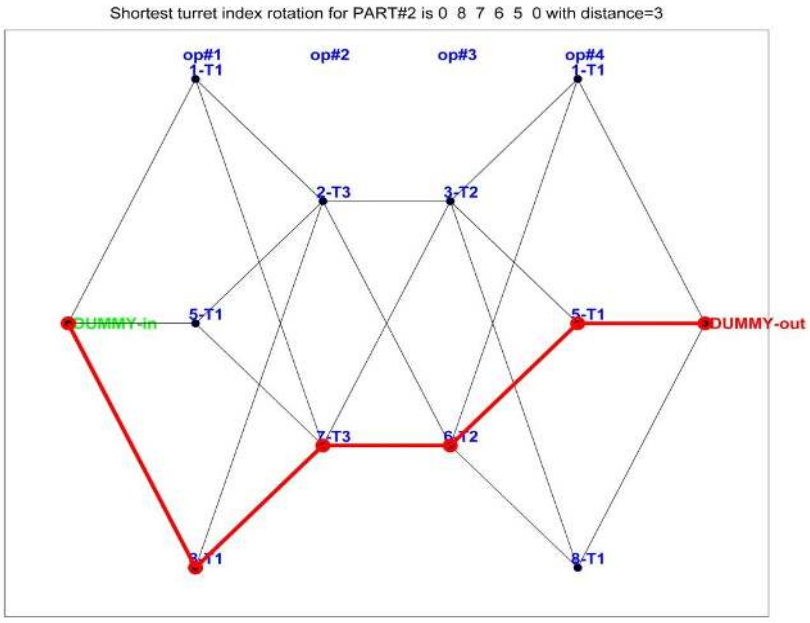


(b)

Figure 6.9 Cutting stages and ATC indexing calculation networks for a) PART3, b) PART1, c) PART4 and d) PART2



(c)



(d)

Figure 6.9 Cutting stages and ATC indexing calculation networks for a) PART3, b) PART1, c) PART4 and d) PART2 (continue)

- No cost is incurred for tools in the magazine.
- Each tool is assumed to fit in one slot of the magazine.

- The required time to remove or to load a tool is constant and is identical for all tools.
- Tools cannot be switched simultaneously.
- The subset of tools required to perform each job is fixed in advance.
- Tools do not break down and do not wear out.
- The list of jobs is completely known.

Under these considerations, if a different tool type is required on any turret position for the next part, it means that the current tool is removed from that position and the required tool type is loaded on the same position. It yields to two switches (movements) (Figure 6.10.a, the *1*st position of the turret). In another case, any position might already be loaded with any tool type and according to the next tool allocation policy of the next part; this position is left empty (Figure 6.10.a, the *4*th position of the turret). Thus, only the mentioned tool is removed from its current position and it makes one switch. Similarly, any position might currently be empty and that position requires a tool for the next part (Figure 6.10.b, the *4*th position of the turret). This move makes one switch again. Finally, if the required tool types are same for the current and for the next part of any position on the turret, a switch is not performed (Figure 6.10.c, the *2*nd position of the turret).

Finally, one can see form Figure 6.10 that, 11, 13 and 8 tool switches are required while passing from PART1 to PART2, from PART2 to PART3 and from PART3 to PART4, respectively. In this example, according to the given tool allocation polices in Figure 6.6, 8, 14 and 13 tool switches are required while passing from PART3 to PART1, from PART1 to PART4 and from PART4 to PART2, respectively. It is assumed that each switch can take 1, 5, 10 and 15 seconds for a defined problem instance. In the present example, switching a tool takes 5 seconds. Finally, by summation of total ATC indexing time and total tool switching time, it is concluded that $145 + 5 \times (8 + 14 + 13) = 320$ seconds of non-machining time is required to complete the whole production.

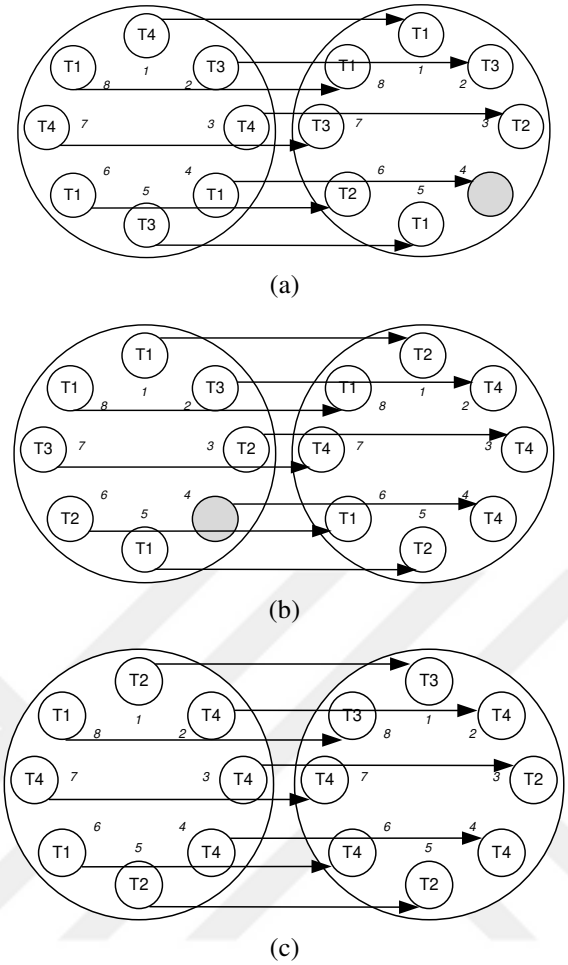


Figure 6.10 Tool switches between a) PART1 and PART2, b) PART2 and PART3 and c) PART3 and PART4.

6.3 The Proposed Solution Approach

6.3.1 SA Algorithm

SA is a stochastic neighborhood search technique that is devised for combinatorial optimization problems. SA is introduced by Kirkpatrick, Gelatt & Vecchi (1983) based on a metaphor between the process of annealing and search in combinatorial optimization problems. Through the search process of SA, improving solutions are directly accepted, whereas the non-improving solutions are accepted if a probability represented by a metaphor of a control parameter (temperature) is satisfied. According to this metaphor, as the temperature is decreased in annealing

process, the probability of accepting non-improving solutions is decreased as well. Thus, SA has the capability of escaping from local optima traps.

6.3.2 Acceptance in SA

In SA, an improving solution is directly accepted whereas a non-improving solution can only be accepted if the acceptance probability is satisfied. The mentioned probability allows a larger portion of the non-improving solutions to be accepted at the initial steps of SA; however, this probability is decremented towards the end of the run depending on systematic decrease of the temperature, which results in accepting severely fewer non-improving solutions at the final steps of SA.

Assume that *(S, f)* is an instant of the problem and *i* and *j* are two solutions with objective function values *f(i)* and *f(j)*, respectively. Then, given in Equation 6.1, the acceptance probability of *j*, a neighboring solution of *i* is determined as follows:

$$P_{a\{j\}} \begin{cases} 1 & if \quad f_{(j)} \leq f_{(i)} \\ e^{-\left(\frac{f_{(j)}-f_{(i)}}{T}\right)} & if \quad f_{(j)} > f_{(i)} \end{cases} \tag{6.1}$$

where $T \in \mathbb{R}^{+}$and P_a represent the temperature and probability of acceptance of the generated solution, respectively.

6.3.3 Input Configuration and Cooling Schedule of SA

In the present work, the user defined input parameters and cooling schedule employed in SA is adopted from and Baykasoglu & Dereli (2004) and Baykasoğlu & Gindy (2001). As mentioned before SA algorithm is a metaphor based on the annealing process of solids. Therefore, it requires an initial temperature first. Setting the initial temperature at very high levels, results in a high degree of randomness at the initial stages of the search (Lin, Kao & Hsu, 1993). Initial value of temperature (T_{in}), can be modified in such a way that at the initial stages of SA, the value of acceptance probability (P_a) is close to unity. In the proposed SA, initial probability of acceptance (P_{in}) is assumed to be equal to 0.95, where T_{in} can be evaluated as in Equation 6.2.

$$T_{in} = (f_{min} - f_{max}) / \ln P_{in} \tag{6.2}$$

where, f_{min} and f_{max} represent the lower and higher bounds for a given problem, respectively. Those values can approximately be estimated by making a number of trial runs.

Another parameter used in SA is the length of inner loop *(LMC)*, where a local search is performed at a stable temperature. However, at the initial stages of SA, because P_a is close to unity, this search might go infinity. In order to prevent this circumstance, in the present work, *LMC* is restricted by a limiting number, which is defined as given in Equation 6.3.

$$LMC = 2 \times posNr \tag{6.3}$$

where $posNr$ represents the number of slots on turret magazine. For the ATCs with 16 slots, *LMC* is fixed to 20.

Cooling rate *(α)* used in the proposed approach is an important factor affecting the success of the algorithm. As given in Equation 6.4, $\alpha \in (0,1)$ is the rate of the decrement in temperature, where T_k is the temperature of the *k*th iteration. Thus, the temperature of the next iteration (T_{k+1}) and depending on this value, the probability of acceptance in Equation 6.1 is regulated.

$$T_{k+1} = \alpha \times T_k \tag{6.4}$$

If α is fixed very close to zero and the temperature is decreased severely fast. On the other hand, if it is chosen very close to unity and the temperature is decreased very slowly, where the algorithm might behave as a random search. Therefore, a precise balance is required here. The definition of Bennage & Dhingra (1995) for cooling rate is adopted here. Thus, α is evaluated as given in Equation 6.5.

$$\alpha = \left(\ln P_{in} / \ln P_f\right)^{(it_{max}-1)^{-1}} \tag{6.5}$$

where P_{in}, P_f and it_{max} represent the initial probability of acceptance, final probability of acceptance and the maximum number of iterations, respectively. As

one can estimate, P_a approaches to zero at the final stages of SA. The minimum value of P_a is referred as P_f and is fixed to 1×10^{-15}. It is clear that values of these probabilities can be sorted as $P_f \leq P_a \leq P_{in}$. Finally, the termination criterion is defined as reaching the final temperature (freezing temperature) or similarly performing it_{max} iterations as well. The value of final temperature (T_f) at the end of the last iteration can be calculated by using Equation 6.6.

$$T_f = T_{in}\alpha^{it_{max}} \tag{6.6}$$

If T_f is not very close or equal to the estimated final temperature then it_{max} must be reselected and the rate of the cooling (α) should be recomputed. This procedure should be repeated until it_{max} is determined.

A pseudo code summarizing the whole process implemented in SA algorithm is depicted in Figure 6.11.

```
//Load test problem; configure input parameters LMC, P_in, P_f, T_in, T_f, it_max, α;
x=genSol()  //a random or known solution is generated;
f(x)=dijkstra(x)+setupEval(x) //setupEval calculates the total switches between parts;
incumbent_sol=x;
incumbent=f(x);
while itr≤ it_max
        while LMC not reached
                x'=genNeigh(x)// neighborhood is generated using current solution;
                f(x')=dijkstra(x')+setupEval(x');
                if f(x')≤f(x) or acceptance criterion is met
                        x=x';
                        f(x)=f(x');
                        if f(x)≤ incumbent
                                incumbent_sol=x;
                                incumbent=f(x);
                        end if
                end if
        end while
        T=T×α;
        itr++;
end while
print(incumbent_sol, incumbent);
```

Figure 6.11 A pseudo code for SA

6.4 Experimental Study in Static Environment

Literature does not include a benchmarking set for the present problem. Therefore, the benchmarking data introduced in Baykasoğlu & Ozsoydan (2016),

which is generated for the static version of a single part ATC indexing problem is expanded here (Appendix 1) to define multiple parts along with their lot sizes and new lower bounds.

In Appendix-1, the columns "*number of operations*" and "*ID of the problem*" represent the total number of operations for the corresponding part, for which the process plans are already given in Baykasoğlu & Ozsoydan (2016) (Appendix 2) and the index of the related problem, respectively. As one can see from this table, there exist 30 different parts each with different process plans. The columns "*LS*" denote the lot sizes of each of the parties with same type of parts. Because the introduced problem is frequently encountered in FMSs, it is assumed that relatively small or medium sized batches rather than thousands of jobs is more adequate, therefore, for each part, the values of *LS* are generated as random integers between [10,15] for benchmarking. Finally, the columns "*ATC indexing LB*" indicate the lower bounds of ATC indexing time for the corresponding parts. These values are evaluated as proposed in Baykasoğlu & Dereli (2004).

Let's consider the *1*st part with 30 operations. Assuming that there exist enough tool copies and slots on ATC, the turret can be rotated to one adjacent slot during each rotation. Thus, machining operations of this part can be completed with minimum 26 rotations because there are 27 consecutive cutting stages each requiring different tools. Finally, multiplying it with lot size of this part (26×12), *LB* value of the *1*st part with 30 operations is obtained (Appendix 1). It is assumed that each rotation takes one second and it is clear that these values are not the lower bounds of the introduced problem. These values are later used in lower bound evaluation of the master problem.

In the introduced problem, it is assumed that there are 16 index positions on the ATC and each tool has three available copies. For performance analysis, randomly chosen parts from Baykasoğlu & Ozsoydan (2016) are systematically used here. For example, for the benchmark#1 in Table 6.1, there exist three different types of parts (three parties) in the problem environment. In this benchmark, the *1*st part of 50-operations parts, the *3*rd part of the 30-operations parts and the *5*th part of the 20-

operations parts are randomly chosen. Thus, it assured that any benchmarking problem includes equal number of part types from different sets (50 operations, 30 operations and 20 operations). In a relatively more difficult problem, in benchmark#3, there are nine parties to schedule and to allocate cutting tools on the turret magazine.

The columns "*LB*", "*avg.*", "*best*" and "*gap%*" in Table 6.1 represent the lower bounds for each of the corresponding benchmarking problem, averages of the best found solutions of three independent runs, the best of the best found solutions of three independent runs, and percentage gap between *LB* and the *best*, which is evaluated as $\frac{100\times(best-LB)}{LB}$, respectively. The column "*switching*" represents the unit tool loading/removing time for the corresponding problem. Because this unit time varies according to the used apparatus by the employees, several different values are tested here. However, it is crucial that switching times should not dominate ATC indexing times, or vice versa. The rest of the columns of Table 6.1 represent the problem sets, which are classified according to the number of operations and the required tool types.

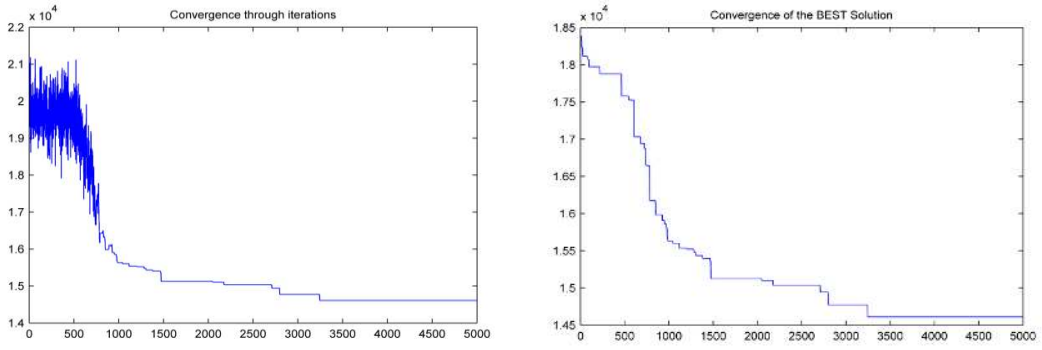
The *LB* values presented in Table 6.1 are obtained by further evaluations using the *ATC indexing LB*s of Appendix 1. Let' s consider the benchmark#1 in Table 6.1. As mentioned earlier, assuming that there is infinite number of tool copies and slots on ATC, *ATC indexing LB*s for the batches, comprised of the *1*st part of 50-operations parts, the *3*rd part of the 30-operations parts and the *5*th part of the 20-operations parts are 410, 405 and 209 seconds, respectively. The final *LB* values in Table 6.1 are simply summation of these values. Thus, assuming that tool switching between parts does not take time, the whole production can be completed in at least 1024 seconds. Similarly, *LB* value for the benchmark#2 is obtained by the sum of 516, 572, 405, 312, 198 and 247.

It is clear that those *LB* values are determined under two assumptions as mentioned above. For this reason, although it is not a tight and practical lower bound technique, it fortunately provides some clues about quality of the found solutions.

The proposed algorithm is coded by using MATLAB, on a PC with hardware of 2.40 GHz processor and 16 GB RAM. The obtained results are presented in Table 6.1.

As one can see from the *gap%* columns, percentage gap values increase as the number of parts and unit tool switching times increase. This is an expected circumstance because the complexity of the introduced problem increases as the number of types of the parts to be processed increases. Additionally, tool switching time is not included in the lower bound technique as given above. Thus, percentage gap values might sometimes increase up to approximately between 200 and 250 in some benchmarking problems. However, this does not directly reflect the performance of the proposed solution approach.

It is clear that benchmark problems 1, 6, 11 and 16 for each of the different switching unit time sections are equivalent. This is also valid for benchmark problems 2, 7, 12 and 17 or similarly for others. As one can see from the corresponding *gap%* values for each of these equivalent problems, similar percentage gap values are obtained by the proposed algorithm. This might provide some positive clues about the robustness of the employed solution approach. A parallel finding is revealed in convergence plots presented in Figure 6.12. As expected, a larger portion of the non-improving solutions is accepted at the initial steps of SA, whereas it is clear that only improving solutions are accepted through the end of the run.



(a) (b)

Figure 6.12 Convergence graph for a) each discovered solution b) the best solution

Table 6.1 Experimental results

switching		*50 oper.*	*30 oper.*	*20 oper.*	*LB*	*avg.*	*best*	*gap%*		*50 oper.*	*30 oper.*	*20 oper.*	*LB*	*avg.*	*best*	*gap%*
1sec.	#1	1	3	5	1024	1786	1770	72.85	#11	7	2	4	1058	1798.33	1771	67.39
	#2	2,4	3,1	9,8	2250	4440.33	4309	91.51	#12	4,9	1,5	2,8	2348	4854	4778	103.49
	#3	1,7,3	10,7,3	2,8,3	3142	6983.33	6910	119.92	#13	4,1,7	5,8,2	4,1,10	2973	6416.33	6285	111.40
	#4	7,9,4,6	7,9,3,2	10,2,5,4	4496	10758.33	10494	133.41	#14	9,2,5,4	3,1,4,9	2,10,7,6	4595	10562	10534	129.25
	#5	3,6,4,5,1	3,2,8,4,7	3,5,4,10,1	5396	13320	13182	144.29	#15	10,3,4,7,1	5,3,8,10,2	3,5,7,4,9	5243	12830.33	12677	141.79
	#6	5	7	2	1298	2398	2340	80.28	#16	4	8	3	1112	1928	1887	69.69
	#7	3,6	1,4	6,8	2095	4132.33	4080	94.75	#17	10,6	3,5	1,6	2178	4501.67	4418	102.85
	#8	1,4,2	6,3,2	8,10,9	3215	7131.33	7010	118.04	#18	2,8,5	1,6,4	2,7,3	3553	8075	8003	125.25
	#9	7,1,3,6	5,9,3,4	7,1,10,2	4010	9357	9298	131.87	#19	4,6,1,3	5,8,10,2	3,7,2,9	4088	9595.33	9532	133.17
	#10	8,10,2,4,1	5,3,9,8,10	4,1,7,3,2	5354	13131.67	12872	140.42	#20	5,7,4,6,2	3,8,9,4,1	2,4,1,9,10	5498	13056.33	12812	133.03
5sec.	#1	1	3	5	1024	2023	1995	94.82	#11	7	2	4	1058	2007	1981	87.24
	#2	2,4	3,1	9,8	2250	4993.33	4898	117.69	#12	4,9	1,5	2,8	2348	5503.33	5428	131.18
	#3	1,7,3	10,7,3	2,8,3	3142	7900.33	7739	146.31	#13	4,1,7	5,8,2	4,1,10	2973	7352	7155	140.67
	#4	7,9,4,6	7,9,3,2	10,2,5,4	4496	11856.33	11759	161.54	#14	9,2,5,4	3,1,4,9	2,10,7,6	4595	12084.33	12040	162.02
	#5	3,6,4,5,1	3,2,8,4,7	3,5,4,10,1	5396	14840.67	14610	170.76	#15	10,3,4,7,1	5,3,8,10,2	3,5,7,4,9	5243	14384.33	14170	170.27
	#6	5	7	2	1298	2608.67	2559	97.15	#16	4	8	3	1112	2140	2062	85.43
	#7	3,6	1,4	6,8	2095	4865	4785	128.40	#17	10,6	3,5	1,6	2178	5070.67	4977	128.51
	#8	1,4,2	6,3,2	8,10,9	3215	7965	7807	142.83	#18	2,8,5	1,6,4	2,7,3	3553	9088.67	8966	152.35
	#9	7,1,3,6	5,9,3,4	7,1,10,2	4010	10663.67	10577	163.77	#19	4,6,1,3	5,8,10,2	3,7,2,9	4088	10896	10758	163.16
	#10	8,10,2,4,1	5,3,9,8,10	4,1,7,3,2	5354	14664.67	14405	169.05	#20	5,7,4,6,2	3,8,9,4,1	2,4,1,9,10	5498	14762.33	14538	164.42
10sec	#1	1	3	5	1024	2298.33	2251	119.82	#11	7	2	4	1058	2201.33	2172	105.29
	#2	2,4	3,1	9,8	2250	5807.33	5757	155.87	#12	4,9	1,5	2,8	2348	6287.33	6149	161.88
	#3	1,7,3	10,7,3	2,8,3	3142	9018.67	8930	184.21	#13	4,1,7	5,8,2	4,1,10	2973	8554.67	8362	181.26
	#4	7,9,4,6	7,9,3,2	10,2,5,4	4496	13416.67	13310	196.04	#14	9,2,5,4	3,1,4,9	2,10,7,6	4595	13605	13558	195.06
	#5	3,6,4,5,1	3,2,8,4,7	3,5,4,10,1	5396	16898	16538	206.49	#15	10,3,4,7,1	5,3,8,10,2	3,5,7,4,9	5243	16198.67	16032	205.78
	#6	5	7	2	1298	2847.67	2699	107.94	#16	4	8	3	1112	2400	2367	112.86
	#7	3,6	1,4	6,8	2095	5649.33	5585	166.59	#17	10,6	3,5	1,6	2178	5899.33	5858	168.96
	#8	1,4,2	6,3,2	8,10,9	3215	9135.33	9068	182.05	#18	2,8,5	1,6,4	2,7,3	3553	10078	10014	181.85
	#9	7,1,3,6	5,9,3,4	7,1,10,2	4010	12207	12066	200.90	#19	4,6,1,3	5,8,10,2	3,7,2,9	4088	12392.33	12284	200.49
	#10	8,10,2,4,1	5,3,9,8,10	4,1,7,3,2	5354	16760	16688	211.69	#20	5,7,4,6,2	3,8,9,4,1	2,4,1,9,10	5498	16724.67	16484	199.82
15sec	#1	1	3	5	1024	2478	2446	138.87	#11	7	2	4	1058	2393.67	2378	124.76
	#2	2,4	3,1	9,8	2250	6644	6512	189.42	#12	4,9	1,5	2,8	2348	7096.67	6973	196.98
	#3	1,7,3	10,7,3	2,8,3	3142	10176.67	10050	219.86	#13	4,1,7	5,8,2	4,1,10	2973	9750.67	9727	227.18
	#4	7,9,4,6	7,9,3,2	10,2,5,4	4496	15071	14846	230.20	#14	9,2,5,4	3,1,4,9	2,10,7,6	4595	15241.33	15053	227.60
	#5	3,6,4,5,1	3,2,8,4,7	3,5,4,10,1	5396	19061.67	18934	250.89	#15	10,3,4,7,1	5,3,8,10,2	3,5,7,4,9	5243	18491	18467	252.22
	#6	5	7	2	1298	3014.33	2900	123.42	#16	4	8	3	1112	2575	2538	128.24
	#7	3,6	1,4	6,8	2095	6261	6207	196.28	#17	10,6	3,5	1,6	2178	6528	6452	196.24
	#8	1,4,2	6,3,2	8,10,9	3215	10404.33	10269	219.41	#18	2,8,5	1,6,4	2,7,3	3553	11580.33	11457	222.46
	#9	7,1,3,6	5,9,3,4	7,1,10,2	4010	13699.33	13512	236.96	#19	4,6,1,3	5,8,10,2	3,7,2,9	4088	14078.33	13880	239.53
	#10	8,10,2,4,1	5,3,9,8,10	4,1,7,3,2	5354	18732	18619	247.76	#20	5,7,4,6,2	3,8,9,4,1	2,4,1,9,10	5498	18976	18713	240.36

6.5 Experimental Study in Dynamic Environment

In FMSs, lot sizes of the batches might change during the ongoing production. Customer might demand for additional parties or simply they can cancel a part of an already given orders. Additionally, in the introduced problem, unit tool switching time can change because it is assumed to be manually performed by the employees using some particular apparatus. Using these components require specific skills and experience, which can vary from an employee to another. It is clear that an employee can be tasked in different machining centers. Thus, it can be concluded that unit tool switching time can change when the responsible employee of a machining centre changes. Employees can be rotated to different machining centers due to several reasons throughout an ongoing production. However, unit ATC indexing time cannot change because it is a human-free automatic machine.

In this dissertation, it is assumed that new production plans, including new lot sizes are given to the corresponding machining centers when a dynamic change in lot

sizes is triggered. Similarly, to changes in lot sizes, rotation of employees can occur at any time during production. However, for a better visualization and comparison of the algorithms here, the frequencies of these two non-dimensional changes are assumed to be equal. In other words, it is assumed that these two changes happen at the same time. However, one can optionally use these changes with different frequencies or randomly at any time in time axis. To sum up, the mentioned problem includes periodically triggered non-dimensional dynamic changes arising from changes in lot sizes and changes in unit tool switching times.

In order the handle the above mentioned problem, four different strategies are proposed here. The first strategy is exactly the same SA, used in the static version of this problem with a slight difference that when a dynamic event is triggered, the temperature is increased up to its initial level. The second strategy further utilizes a reinitializing mechanism in the canonical SA. In this technique (SA-reinit), additionally to the base SA, whenever a dynamic event is triggered, the algorithm reinitializes itself with its initial configuration. Therefore, SA indeed always uses the best found solution of the current shift, as the initial solution of the next shift. It tries to further improve that solution with new parameters of the new environment. On other hand, SA-reinit starts the optimization process of the new shift with a randomly generated new solution. The algorithms without reinitializing procedures are referred as *robust strategies* in this dissertation.

6.5.1 Multiple Start SA

Multiple Start SA (msSA) (Boese, Kahng & Muddu, 1994; Lin, 2013; Lin, Ying, Lu & Gupta, 2011; Van Hentenryck & Vergados, 2007) is an extension of the canonical SA. The aim in msSA is to provide a more diversified search, which is actually a parallel idea with introducing diversity based methods, widely used in dynamic optimization community (Chapter 3).

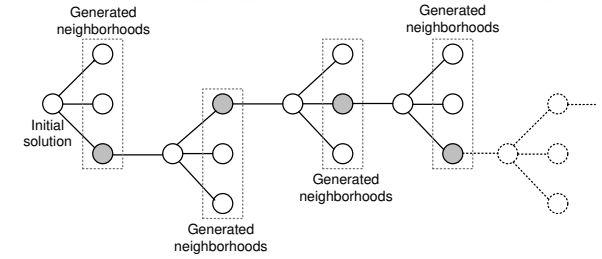
Similar to SA, msSA starts with a random initial solution, but it performs an additional intensification of the current solution just before proceeding to inner loop neighborhood search stage. Therefore, denoted by *outer_trial*, first a number of neighborhood solutions are randomly generated using the current solution. Next, the

generated solutions are ranked in ascending order w.r.t. their fitness values. The first rank is the fittest solution. Afterwards, probabilities of each ranked solution are evaluated as formulated in Equation 6.7. The found probabilities are normalized by using Equation 6.8. Next, a random number is generated and using a roulette wheel procedure with the normalized probabilities, a random solution out of *outer_trial* solutions is chosen. Finally, this randomly chosen solution is compared with the current solution, which is indeed the parent of the generated *outer_trial* solutions. Simply if the condition formulated in Equation 6.1 is satisfied, the current outer solution is replaced with this new solution. Otherwise no actions are performed.

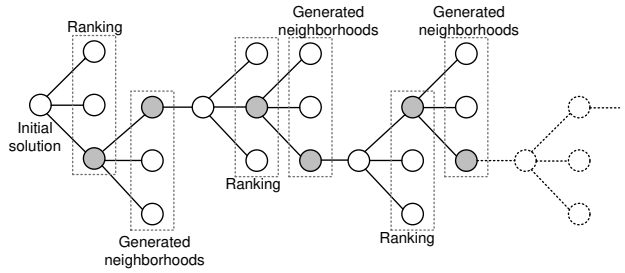
An illustration of the strategies of SA and msSA are presented in Figures 6.13.a and 6.13.b, respectively. Additionally a pseudo code for msSA is provided in Figure 6.14.

$$prob_i = \beta_{ranking} \times rank_i{}^{-\alpha_{ranking}} \qquad \forall i \tag{6.7}$$

$$normalized_i = {prob_i}\Big/{\sum_{j=1}^{outer\_tiral} prob_j} \qquad \forall i \tag{6.8}$$



(a)



(b)

Figure 6.13 a) Search of SA b) search of msSA

```
//Load test problem; configure input parameters LMC, P_in, P_f, T_in, T_f, it_max, α, α_ranking, β_ranking
x=genSol()  //a random or known solution is generated;
f(x)=dijkstra(x)+setupEval(x) //setupEval calculates the total switches between parts;
incumbent_sol=x;
incumbent=f(x);
while itr≤ it_max
        x'=genNeigh(x) // neighborhood solutions are generated using current solution;
        if min{f(x')} ≤ incumbent
                incumbent_sol=x';
                incumbent=f(x');
        end if
        rank x';
        evaluate probabilities of each ranked solution x' via Equation 6.7
        normalize & select x'' by roulette wheel selection
        if f(x'') ≤ f(x) or acceptance criterion is met
                x=x'';
                f(x)= f(x'');
        end if
        while LMC not reached
                x'=genNeigh(x)
                f(x')=dijkstra(x')+setupEval(x');
                if f(x') ≤ f(x) or acceptance criterion is met
                        x=x';
                        f(x)= f(x');
                        if f(x) ≤ incumbent
                                incumbent_sol=x;
                                incumbent= f(x);
                        end if
                end if
        end while
        T=T × α;
        itr++;
end while
print(incumbent_sol, incumbent);
```

Figure 6.14 A pseudo code for msSA

In comparison to canonical SA, msSA has additionally three more parameters denoted by *outer_trial*, $\alpha_{ranking}$ and $\beta_{ranking}$. The first parameter *outer_trial* can arbitrarily be fixed to any level. However, a greater *outer_trial* might provide more qualified solutions with the trade of more evaluations. The other two parameters $\alpha_{ranking}$ and $\beta_{ranking}$ tune severity of probability decrement in ranking. The main idea adopted here is to give approximately 50% chance to the first 25% leading solutions out of *outer_trial* solutions. Therefore, *outer_trial*, $\alpha_{ranking}$ and $\beta_{ranking}$ are fixed to 10, 0.99 and 0.98 throughout the experimental study in dynamic environments, respectively.

The final method used in dynamic environments is simply the reinitializing procedure added modification of msSA. This procedure is denoted as msSA-reinit in this thesis. The relation between msSA and msSA-reinit is parallel to the relation between SA and SA-reinit. Whenever a dynamic event is triggered, the temperature

is increased up to its initial level in all used algorithms, SA, SA-reinit, msSA and msSA-reinit.

6.5.2 Features of the Dynamic Events in the Introduced Problem

In the dynamic extension of the introduced problem, two non-dimensional changes are allowed. The former one is changes in lot sizes. This parameter is assumed to change as random integers between [10, 15]. The latter one is unit tool switching time, which is dependent to skill and experience of the employee of the corresponding shift. This parameter is assumed to change as random integers between [5, 10].

A dynamic event detection mechanism is not necessarily required here because it is assumed that new problem environments are revealed to the algorithm because the new production plans are always given to the related machining centers when a dynamic event is triggered. However, the new parameters of the upcoming environments remain unknown until the new orders are received. Therefore, the question is whether to restart the optimization from scratch with the new parameters or to use and to adapt the best-found solution of the previous environment to the new environment. Although, generally speaking, frequency of the changes are hard to predict, in this thesis, as 500 (high frequency), 1000 (medium frequency) and 2500 (low frequency), three different shift lengths are used in order to present different levels of dynamism.

6.5.3 Experimental Results in Dynamic Environments

Finding global optimum of each environment for the introduced problem is not an easy task. Therefore, rather than using error and optimality based performance measures of dynamic optimization, best-of-generations (Equation 2.9) and best-before-change (Equation 2.10) are adopted here. Because SA is not a population based approach, the best solution of a generation is assumed as the best-found solution until that generation. It is identical to using this performance measure for evaluating the best-of-generations performance of a population based algorithm with an elitist strategy. Additionally, Equation 2.10 uses the best error values before

dynamic changes. Here, this value is used as the best-found solution just before changes. Best-of-generations and best-before-change performances of the used algorithms are evaluated over 10 independent runs and are presented in Tables 6.2 and 6.3, respectively.

Table 6.2 Best-of-generations performances of the algorithms under varying frequencies

	frequency=500	*frequency=1000*	*frequency=2500*
SA	16513.33	16319.72	16093.50
SA-reinit	17527.35	17354.46	17164.94
msSA	16493.69	16280.43	16004.51
msSA-reinit	17460.81	17307.48	17065.95

Table 6.3 Best-before-change (standard error) performances of the algorithms under varying frequencies

	frequency=500	*frequency=1000*	*frequency=2500*
SA	16294.86 (204.24)	16089.76 (195.32)	15861.99 (199.39)
SA-reinit	16398.61 (209.35)	16150.75 (209.98)	15943.54 (210.46)
msSA	16273.06 (208,78)	16066.48 (191.55)	15796.69 (198.77)
msSA-reinit	16294.86 (204.24)	16089.76 (195.32)	15861.99 (199.39)

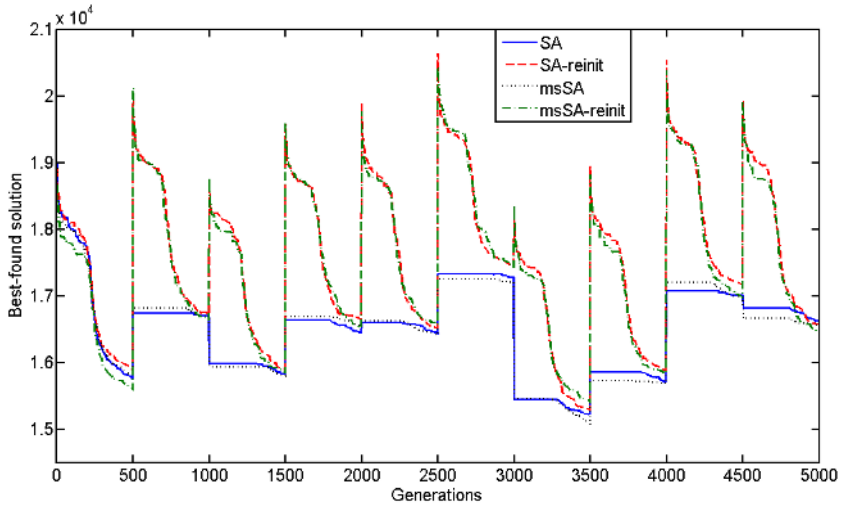
One surprising result is apparent in Tables 6.2 and 6.3. In many DOPs, introducing diversity based methods yield to a better performance particularly in evolutionary algorithms. A parallel strategy is also adopted in SA-reinit and msSA-reinit. Comparing SA with SA-reinit and msSA with msSA-reinit, one can see that this strategy does not contribute to the performance of the algorithms. On the other hand, using the best-found solution of the current environment in the next environment and trying to further improve that solution, yields to better results. It is worth stressing that, under more severe changes including dimensional changes, reinitializing procedure could be the more promising one but with the present experimental suite, a decision maker can conclude to use the robust strategy.

Compared to SA, msSA performs *outer_trial* more evaluations at each generation. Therefore, it clear that msSA is expected to find better solutions. As one can see from Tables 6.2 and 6.3, this is achieved by msSA. This is also valid while comparing SA-reinit with msSA-reinit.

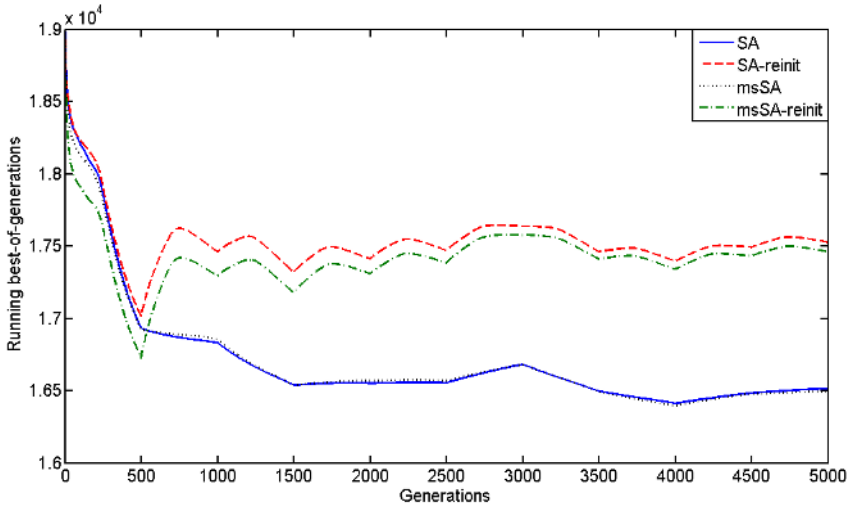
Best-found solution values and running-best-of-generations for each of the frequency levels are presented in Figures 6.15, 6.16 and 6.17. According to Figures 6.15.b, 6.16.b and 6.17.b, where the overall behavior of each corresponding algorithm are plotted, msSA-reinit clearly outperforms SA-reinit. Due to its additional evaluations before inner neighborhood generation loop, it searches for additional neighboring solutions. Therefore, it has more chance in finding more promising solutions in shorter generations. Or similarly, if it loses the track of optima, there is a greater possibility for a faster recovery for msSA-reinit in comparison to SA-reinit.

As a parallel finding to the results of Tables 6.2 and 6.3, one can see from the same figures that, reinitializing procedure does not contribute to the solution quality in this problem. Particularly in high frequency environment (500), the gap between the reinitializing and the robust strategy increases. The differences between all four strategies are more clear in Figure 6.17.b, where the environments are allowed to remain stationary for a longer period. As one can see from this figure, the lines of robust strategies are always below the lines of reinitializing strategies. In addition, it is clear from the same figure that, msSA is always better than SA regardless of the adopted strategy (robust or reinitializing). This is also reflected in Figures 6.15.a, 6.16.a and 6.17.a. It is clear from these figures that reinitializing strategy cannot catch the solutions of robust strategy even at the end of most of the environments.

Best-before-change plots (Figure 6.18) tells us a new thing that is not revealed by Figures 6.15-6.17. One can see from these figures that, each of these algorithms nearly overlap in terms of the best-found solution at the end of the environments. In means that actually the precision of these algorithms are competitive. It is also an expected result because msSA is not a superiorly enhanced extension of canonical SA. What msSA does is only to perform agreed additional evaluations just before inner loop neighborhood search. However, due this additional search, msSA achieves better results in shorter time and this contributes to its overall performance, which is a crucial feature in dynamic optimization schemes. Nevertheless, it can be concluded that their precision are still competitive.

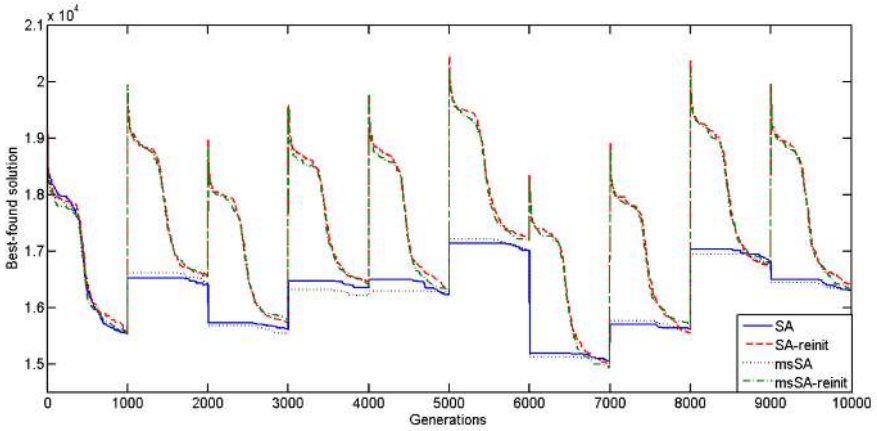


(a)



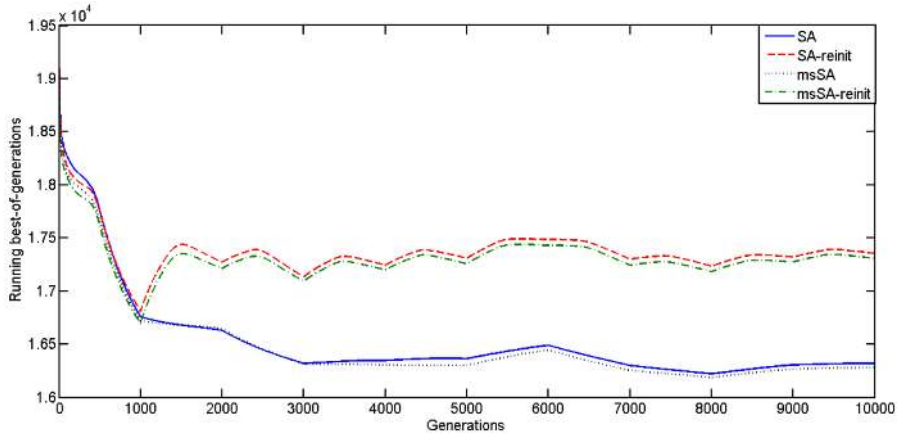
(b)

Figure 6.15 a)Best-found solution b)running best-of-generation values for 500 frequency



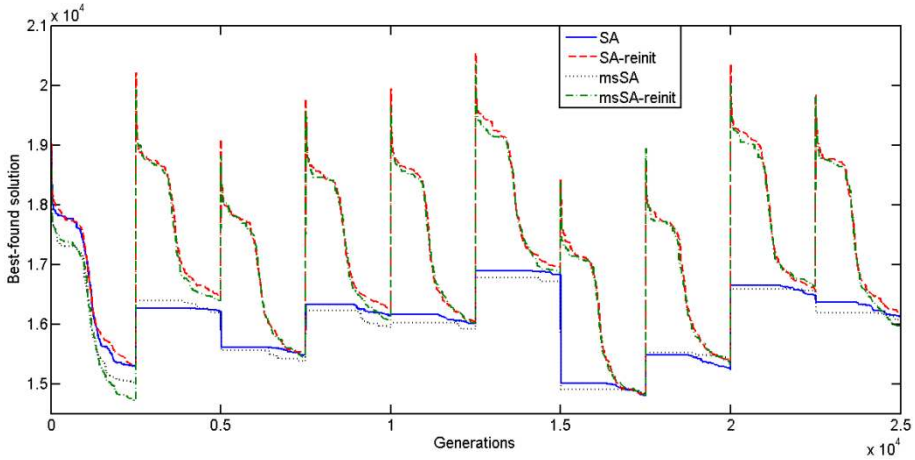
(a)

Figure 6.16 a)Best-found solution b)running best-of-generation for 1000 frequency

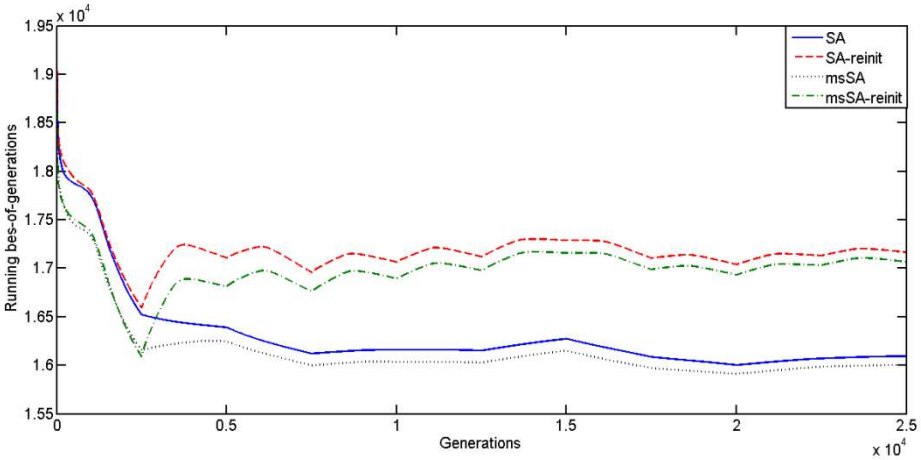


(b)

Figure 6.16 a)Best-found solution b)running best-of-generation for 1000 frequency (continue)

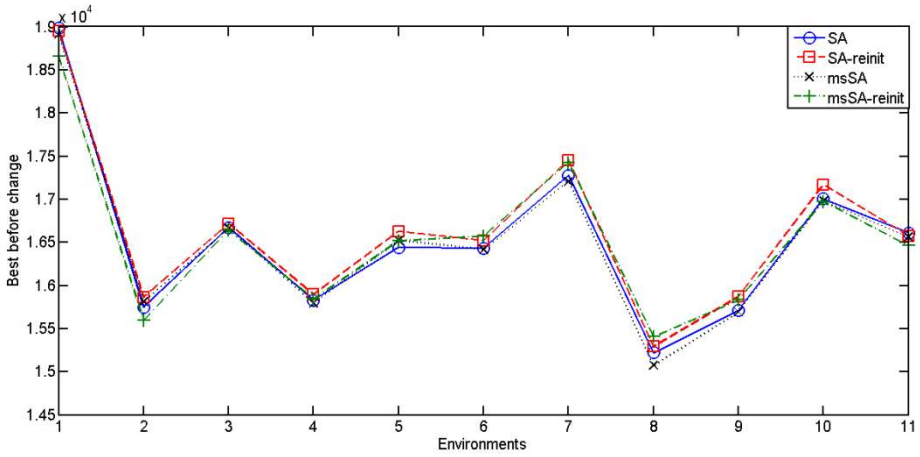


(a)

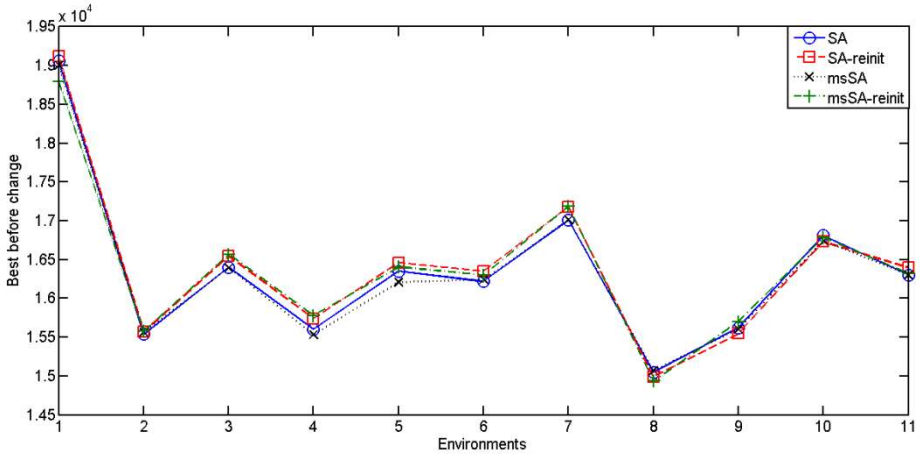


(b)

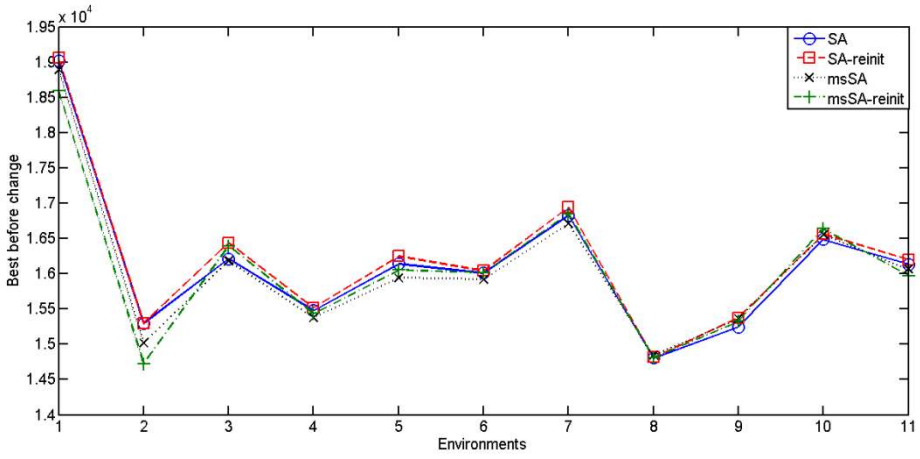
Figure 6.17 a) Best-found solution b) running best-of-generation values for 2500 frequency



(a)



(b)



(c)

Figure 6.18 Best before change plots for a) frequency=500 b) frequency=1000 c) frequency=2500

6.6 Chapter Conclusion

In this chapter, first, two challenging problems in operating automated machining centers namely, automatic tool changer (ATC) indexing problem and tool switching problem (ToSP) are simultaneously handled in order to reduce non-machining times. Distinctly from the previous studies, cutting tool duplications are also considered here. However, this consideration brings a new dimension to the problem. In other words, an ATC has several possible rotation choices while rotating from one slot to another one. A branching at each of these decision points, (the decision to rotate to one of the duplicated tools) generates numerous rotation alternatives (a tree) for a single tool allocation policy on the turret. Therefore, in order to find the shortest rotation for any corresponding allocation, a shortest path problem, which is another standalone optimization problem, is solved optimally on a generated network comprised of cutting stages and branching alternatives. This is one of the unique features of the introduced problem.

Moreover, finding a solution for the present problem requires solving several optimization problems simultaneously, including scheduling on a single machine with sequence-dependent setup times, which is a phenomenon problem in its field. The hierarchy between these problems and the environment of the mentioned problem along with illustrative examples and figures are also provided in this chapter.

A simulated annealing algorithm is developed as the solution approach for the present problem. Benchmark problems are generated by expanding and modifying some previously published problems.

Finally, the introduced problem is extended as a dynamic optimization problem including several non-dimensional changes. In order to deal the challenges of this problem, canonical simulated annealing algorithm is enhanced with employing multiple starting procedure allowing for a superior diversity throughout the search.

In order to provide numerical evidences for possible future work, extensive experimental study for both stationary and dynamic environments are presented in this chapter.

CHAPTER SEVEN
CONCLUSION

7.1 Summary of the Dissertation

This thesis focuses on dynamic optimization problems and their applications in real-life manufacturing systems. Considering the published work so far in dynamic optimization community, one can see that real-life cases and applications of dynamic optimization concepts in industrial manufacturing systems are lacking. Moreover, a great majority of the publications adopt continuous problems, whereas most of the real-life problems are combinatorial in their nature.

In this respect, first, in Chapter 2, some common concepts of dynamic optimization field are presented. This chapter involves introducing several attributes of dynamic optimization problems, commonly used performance measures, benchmarks and benchmark generators.

In the following chapter, a multi-population based algorithm for multi-modal and continuous problems is developed. The aim here is to gain some practical insights to be used in later applications.

In Chapter 4, some of the fundamental evolutionary and population based algorithms are compared to the constructive and multi-start search strategy. The tests are conducted on a dynamic extension of a well-known combinatorial problem, which has numerous applications in real-life systems.

A case study, adopting this strategy, is presented in Chapter 5. The aim of this case study is to generate efficient and applicable schedules to the parallel heat treatment furnaces of a fastener manufacturer company, which is particularly operating in automotive industry. Besides many restrictions and difficulties existing in this problem, one of the most challenging issues is dynamism of the problem. Capturing and handling dynamism of this highly changing problem requires quite a different effort, clarified in details in Chapter 5.

Finally, another example, covering a flexible manufacturing system with tool switching problem, machining operations and cutting tool indexing problem in CNC machine turrets, is presented in Chapter 6. A neighborhood based strategy with an integrated shortest path algorithm is used as a solution approach here. Next, some further modifications of this strategy along with its base version are tested on a dynamic extension of the introduced problem.

7.2 Contributions

Although partitioning whole population into smaller sized sub-populations as in the developed algorithm in Chapter 3 is not new in dynamic optimization field, making use of chaos along with their inverse patterns and Lévy flights while searching for the promising regions, can be considered as a novelty. With the help of chaos and the specialized move procedure, more enhanced diversification capability is also provided to the algorithm. Additionally, making use of infeasible solutions, which is far ignored in the related literature, is also allowed in the proposed algorithm. The merit in using infeasibility is to initialize some of the population out of the boundaries of the problem space, and make them get attracted by some other inbound individuals in order to find the hard-to-detect promising regions particularly near to the boundaries of the problem space. A specialized relocation mechanism to prevent overlapping of the relocated individuals while tracking the already found optima is also employed in the proposed approach.

In Chapter 4, rather than employing evolutionary algorithms, making use of constructive and multi-start search strategy is proposed particularly in combinatorial dynamic optimization problems. Using such a strategy in a time-varying problem has several benefits. First, the loss of diversity problem is no longer an issue of concern, whereas diversity can easily be lost in evolutionary strategies. Second, any explicit response strategy to dynamic events is not necessarily required in the proposed approach. These are the advantages of multi-start strategy. Finally, third, possible deficiencies in solution representations due to dimensional changes are also not issue of concern because the proposed algorithm does not use a fixed length representation. This is the advantage of the constructive strategy. Combining these

two features, it is demonstrated that fast and efficient solutions can easily be found by the proposed approach. However, it is clear that, one needs to put forward an additional effort while using memory based strategies within this method, because of the multi-start feature.

The proposed approach is implemented through a case study, presented in Chapter 5. Since the interested problem is highly dynamic, the aim is to generate efficient and applicable schedules as fast as possible rather than obtaining the best solution. Because, every wasted second here means unnecessary consumption of natural gas and electricity, to increase or to decrease the temperature level of the parallel heat treatment furnaces or to keep it stable as well. It is obvious that evolutionary strategies require to perform some evaluations to find a qualified solution, whereas the proposed strategy has the chance to obtain a promising solution just in the first generation. Thus, by making use of the proposed approach in real-life systems as in the presented case study, considerable capital saving along with the decrement in resource and energy consumption is possible.

Finally, in Chapter 6, first a shortest path algorithm is proposed to find the exact ATC indexing time of a turret, where cutting tool duplications are allowed. This is one of the novelties proposed for this problem. Although with the assumption of tool duplications, the problem becomes harder, it is obvious that, lower ATC indexing times can be achieved. This is surprisingly far ignored in the related literature. Next, it is stated that in order to decrease the total non-machining times in machining centers of flexible manufacturing systems, tool switching and ATC indexing problems should be handled simultaneously, which is also commonly ignored. The proposed approach is tested both in static and dynamic extension of several benchmark instances introduced in this thesis.

7.3 Discussion & Future Research Directions

Throughout the research studies of this dissertation, it has become clear that some issues can possibly contribute to future research. These issues can be classified as presented in the following.

First, it is surprising that the advantages of employing constructive and multi-start search strategies have not been sufficiently debated in dynamic optimization community. Although evolutionary and population based algorithms have notable merits, it is clear that further effort is required to adopt such approaches in dynamic environments. Constructive and multi-start search strategies on the other hand, are notably convenient particularly for combinatorial dynamic problems in terms easiness in implementation. It is clear that easiness is not a tangible aspect so it is difficult to come to a final decision between these two strategies. In the future, more sophisticated evolutionary and population based algorithms to handle with dimensional changes more efficiently might be developed. However, as some further progress is achieved in this research field, it is apparent that adopting constructive and multi-start strategies particularly in combinatorial dynamic optimization problems will become widespread in the near future.

Dynamic optimization is still a developing research field. It is worth stressing that, rather than adopting synthetic and theoretical problems, a greater number of studies, particularly targeting various real-life applications are crucially needed in this discipline. Because, by making use of dynamic optimization concepts along with optimization schemes in real-life manufacturing systems, utilizing the rare and limited resources in a more efficient way, will become possible.

REFERENCES

Abdunnaser, Y. (2006). *Adapting evolutionary approaches for optimization in dynamic environments*. Phd Thesis, University of Waterloo, Waterloo, Ontario, Canada.

Ackley, D. H. (1987). *A connectionist machine for genetic hillclimbing*. Boston: Kluwer Academic Publishers.

Aisopos, F., Tserpes, K., & Varvarigou, T. (2013). Resource management in software as a service using the knapsack problem model. *International Journal of Production Economics*, 141 (2), 465-477.

Akturk, M. S., & Ozkan, S. (2001). Integrated scheduling and tool management in flexible manufacturing systems. *International Journal of Production Research*, 39 (12), 2697-2722.

Alba, E., Luque, G., & Arias, D. (2009). Impact of frequency and severity on non-stationary optimization problems. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, & P. Machado, (Eds.). *Applications of Evolutionary Computing*, (755-761). Berlin Heidelberg: Springer.

Alba, E., Nakib, & Siarry, A. (2013). *Metaheuristics for dynamic optimization*. Berlin: Springer.

Alba, E., & Sarasola, B. (2010). Abc, a new performance tool for algorithms solving dynamic optimization problems. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, Barcelona, 1-7.

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega-International Journal of Management Science*, 27 (2), 219–239.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187 (3), 985–1032.

Argüello, M. F., Bard, J. F., & Yu, G. (1997). A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, 1 (3), 211-228.

Armentano, V. A., & de Franca Filho, M. F. (2007). Minimizing total tardiness in parallel machine scheduling with setup times: an adaptive memory-based GRASP approach. *European Journal of Operational Research*, 183 (1), 100-114.

Arnold, D. V., & Beyer H.-G. (2002). Random dynamics optimum tracking with evolution strategies. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, & J.-L. Fernández-Villacañas, (Eds.). *Parallel Problem Solving from Nature — PPSN VII* (3-12) Berlin: Springer Heidelberg.

Arnold, D. V., & Beyer, H. G. (2006). Optimum tracking with evolution strategies. *Evolutionary Computation*, 14(3), 291-308.

Avcı S, & Akturk, M. S. (1996). Tool magazine arrangement and operations sequencing on CNC machines. *Computers & Operations Research*, 23 (11), 1069-1081.

Baykasoğlu, A. (2012). Design optimization with chaos embedded great deluge algorithm. *Applied Soft Computing*, 12 (3), 1055-1067.

Baykasoğlu, A., & Dereli. T. (2004). Heuristic optimization system for the determination of index positions on CNC magazines with the consideration of cutting tool duplications. *International Journal of Production Research*, 42 (7) 1281-1303.

Baykasoğlu, A, & Durmuşoğlu, Z. D. U. (2011). Dynamic optimization in a dynamic and unpredictable world. *In Proceedings of Technology Management in the Energy-smart World (PICMET)*, Portland, OR, USA, 2312–2319.

Baykasoğlu, A, & Durmuşoğlu, Z. D. U. (2014). A classification scheme for agent based approaches to dynamic optimization. *Artificial Intelligence Review*, 41 (2), 261-286.

Baykasoğlu, A., & Gindy, N. N. Z. (2001). A simulated annealing algorithm for dynamic layout problem. *Computers & Operations Research*, 28 (14), 1403-1426.

Baykasoğlu, A., Kaplanoğlu, V., Erol, R., & Şahin, C. (2011). A multi-agent framework for load consolidation in logistics. *Transport*, 26 (3), 320-328.

Baykasoğlu, A., & Ozsoydan, F. B. (2014). An improved firefly algorithm for solving dynamic multidimensional knapsack problems. *Expert Systems with Applications*, 41 (8), 3712-3725.

Baykasoğlu, A., & Ozsoydan, F. B. (2015a). Adaptive firefly algorithm with chaos for mechanical design optimization problems. *Applied Soft Computing*, 36, 152-164.

Baykasoglu, A., & Ozsoydan, F. B. (2015b). A GRASP based approach to dynamic scheduling of parallel heat treatment furnaces in a manufacturing company. *In Proceedings of Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA)*, Prague, 656-658.

Baykasoğlu, A, & Ozsoydan, F. B. (2016). An improved approach for determination of index positions on CNC magazines with cutting tool duplications by integrating shortest path algorithm. *International Journal of Production Research* 55 (3), 742-760.

Bennage, W. A, & Dhingra, A. K. (1995). Single and multiobjective structural optimization in discrete-continuous variables using simulated annealing. *International Journal for Numerical Methods in Engineering*, 38 (16) 2753-2773.

Bird, S., & Li, X. (2006). Adaptively choosing niching parameters in a PSO. *In Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, NY, USA, 3-10.

Bird, S., & Li, X. (2007). Using regression to improve local convergence. *In Proceedings of IEEE Congress on Evolutionary Computation*, Singapore, 592-599.

Blackwell, T. M. (2003). Swarms in dynamic environments. In E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska & J. Miller, (Eds.). *Genetic and Evolutionary Computation - GECCO 2003* (1-12). Berlin Heidelberg: Springer.

Blackwell, T. M., & Bentley, P. J. (2002). Dynamic search with charged swarms. *In Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, NY, USA, 9-26.

Blackwell, T., & Branke, J. (2004). Multi-swarm optimization in dynamic environments. In G. Raidl, S. Cagnoni, J. Branke, D. Corne, R. Drechsler, Y. Jin, C. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. Smith & G. Squillero, (Eds.). *Applications of Evolutionary Computing* (489-500). Berlin Heidelberg: Springer.

Blackwell, T., & Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10 (4), 459-472.

Blackwell, T., Branke, J., & Li, X. (2008). Particle swarms for dynamic optimization Problems. In C. Blum & D. Merkle, (Eds.), *Swarm Intelligence* (193-217). Berlin Heidelberg: Springer.

Boese, K. D., Kahng, A. B., & Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16 (2), 101-113.

Bordoloi, S. K, Cooper, W. W, & Matsuo, H. (1999). Flexibility, adaptability, and efficiency in manufacturing systems. *Production and Operations Management*, 8 (2), 133-150.

Boria, N., & Paschos, V. T. (2011). A survey on combinatorial optimization in dynamic environments. *RAIRO-Operations Research*, 45 (03), 241-294.

Branke, J. (1999a). *Evolutionary approaches to dynamic optimization problems-a survey*. Retrieved March 1, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.1576&rep=rep1&type=pdf.

Branke, J. (1999b). Memory enhanced evolutionary algorithms for changing optimization problems. *In Proceedings of Evolutionary Computation Congress (CEC)*, Washington, USA.

Branke, J. (2001a). *Evolutionary approaches to dynamic optimization problems-updated survey*. Retrieved March 1, 2016, from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=336084BE2F2815B4667DC1863ECC364A?doi=10.1.1.10.4619&rep=rep1&type=pdf.

Branke, J. (2001b). *Evolutionary optimization in dynamic environments*. Dordrecht: Kluwer Academic Publishers.

Branke, J., Kaussler, T., Smidt, C., & Schmeck, H. (2000). A multi-population approach to dynamic optimization problems. In I. C. Parmee, (Ed.). *Evolutionary Design and Manufacture* (299-307). London: Springer.

Branke, J, Orbayı, M, & Uyar, Ş. (2006). The role of representations in dynamic knapsack problems. In: F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. H. Moore, J. Romero, G. D. Smith, G. Squillero, & H. Takagi, (Eds.). *Applications of Evolutionary Computing* (764-775). Berlin: Springer-Verlag.

Branke, J., Salihoğlu, E., & Uyar, Ş. (2005). Towards an analysis of dynamic environments. *In Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, NY, USA, 1433-1440.

Branke, J., & Schmeck, H. (2003). Designing evolutionary algorithms for dynamic optimization problems. In A. Ghosh, & S. Tsutsui, (Eds.). *Advances in evolutionary computing* (239-262). Berlin Heidelberg: Springer.

Brest, J., Zumer, V., & Maucec, M. S. (2006). Self-adaptive differential evolution algorithm in constrained real-parameter optimization. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, 215-222.

Brits, R., Engelbrecht, A., & Van Den Bergh, F. (2002). A niching particle swarm optimizer. *In Proceedings of the Conference on Simulated Evolution and Learning*, Singapore, 692-696.

Browne, J., Dubois, D., Rathmill, K., Sethi, S. P., & Stecke, K. E. (1984). Classification of flexible manufacturing systems. *The FMS Magazine*, 2 (2), 114-117.

Buttazzo, G. C., Bertogna, M., & Yao, G. (2013). Limited preemptive scheduling for real-time systems. A survey. *IEEE Transactions on Industrial Informatics*, 9 (1), 3-15.

Cadenas, J. M., Garrido, M. C., & Muñoz, E. (2011). Facing dynamic optimization using a cooperative metaheuristic configured via fuzzy logic and SVMs. *Applied Soft Computing*, 11 (8), 5639-5651.

Campbell, A. M., & Savelsbergh, M. W. (2005). Decision support for consumer direct grocery initiatives. *Transportation Science*, 39 (3), 313-327.

Ceran, E. T. G. (2014). *Dynamic allocation of renewable energy through a stochastic knapsack problem formulation for an access point on the move.* Phd Thesis, Middle East Technical University, Ankara.

Cheng, H., & Yang, S. (2010). Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcazar, C.-K. Goh, J. Merelo, F. Neri, M. Preuß, J. Togelius, & G. Yannakakis, (Eds.).

Applications of Evolutionary Computation (562-571). Berlin Heidelberg: Springer.

Cheng, T. C. E., & Sin, C. C. S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47 (3), 271–292.

Cobb, H. G. (1990). *An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time dependant nonstationary environments*. Technical Report No: ADA229159, Naval Research Laboratory, Washington, USA.

Costa, A., Cappadonna, F. A., & Fichera, S. (2013). A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 69 (9), 2799-2817.

Crama, Y., Kolen, A. W. J, Oerlemans, A. G, & Spieksma, F. C. R. (1994). Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6 (1), 33-54.

Crama, Y., Moonen, L. S., Spieksma, F. C., & Talloen, E. (2007). The tool switching problem revisited. *European Journal of Operational Research*, 182 (2), 952-957.

Cruz, C., González, J., & Pelta, D. (2011). Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15 (7), 1427-1448.

Daas, M. S., & Batouche, M. (2014). Multi-bacterial foraging optimization for dynamic environments. *In Proceedings of International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, Tunis, 237 - 242.

Das, S., & Suganthan, P. N. (2011). Differential evolution: a survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15, 4-31.

Davis, R. I., & Burns, A. (2011). A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43 (4), 35-78.

DeJong, K. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Phd Thesis, University of Michigan, Ann Arbor MI.

Dereli, T., & Baykasoğlu, A. (2005). OPPS-PRI 2.0: An open and optimized process planning system for prismatic parts to improve the performance of SMEs in the machining industry. *International Journal of Production Research*, 43 (5) 1039-1087.

Dereli, T., Baykasoğlu, A., Gindy, N. N. Z., & Filiz, İ. H. (1998). Determination of optimal turret index positions by genetic algorithms. *In Proceedings of International Symposium on Intelligent Manufacturing Systems*, Sakarya, Turkey, 743-750.

Dereli, T., & Filiz, İ. H. (1999). Optimisation of process planning functions by genetic algorithms. *Computers & Industrial Engineering*, 36 (2), 281-308.

Dereli, T., & Filiz, İ. H. (2000). Allocating optimal index positions on tool magazines using genetic algorithms. *Robotics and Autonomous Systems*, 33 (2-3) 155-167.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269-271.

Djellab, H., Djellab, K., & Gourgand, M. (2000). A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, 64 (1-3), 165-176.

Dorigo, M. (1992). *Optimization, learning and natural algorithms*. Phd Thesis, DEI, Politecnico di Milano, Italy.

du Plessis, M. C., & Engelbrecht, A. P. (2012). Using Competitive Population Evaluation in a differential evolution algorithm for dynamic environments. *European Journal of Operational Research*, 218 (1), 7-20.

du Plessis, M., & Engelbrecht, A. (2013). Differential evolution for dynamic environments with unknown numbers of optima. *Journal of Global Optimization*, 55 (1), 73-99.

Du, W., & Li, B. (2008). Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, 178 (15), 3096-3109.

Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, 56 (293), 52–64.

Eberhart, R., & Shi, Y. (2001). Tracking and optimizing dynamic systems with particle swarms. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Seoul, 94–100.

Edis, E. B., & Ozkarahan, I. (2011). A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. *Engineering Optimization*, 43 (2), 135-157.

Elhassania, M., Jaouad, B., & Ahmed, E. A. (2014). Solving the dynamic vehicle routing problem using genetic algorithms. *In Proceedings of Logistics and Operations Management International Conference(GOL)*, Rabat, 62-69.

Erkılıç, T. G. (2014). *Optimizing the service policy of a mobile service provider through competitive online solutions to the 0/1 knapsack problem with dynamic capacity*. Phd Thesis, Middle East Technical University, Ankara.

Erol, R., Sahin, C., Baykasoğlu A, & Kaplanoğlu V. (2012). A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems. *Applied Soft Computing*, 2 (6), 1720-1732.

Feng, W., Brune, T., Chan, L., Chowdhury, M., Kuek, C. K., & Li, Y. (1998). Benchmarks for testing evolutionary algorithms. *In Asia-Pacific Conference on Control and Measurement*, 134-138.

Feng, Y., & Wang, G. G. (2015). An improved hybrid encoding firefly algorithm for randomized time-varying knapsack problems. *In Proceedings of Soft Computing & Machine Intelligence Conference*, Hong Kong, 9-15.

Feo, T., & Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6 (2), 109–133.

Fister, I. Fister Jr. I, Yang, X-S., & Brest, J. (2013). A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*, 13, 34–46.

Fister, I, Yang X-S., Brest, J, & Fister, Jr. I. (2013). Modified firefly algorithm using quaternion representation. *Expert System with Applications*, 40, 7220-7230.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32, 675–701.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics*, 11, 86–92.

Geshlag, M. B. M., & Sheykhzadeh, J. (2013). A new particle swarm optimization model based on learning automata using deluge algorithm for dynamic environments. *Journal of Basic and Applied Scientific Research*, 3 (8), 394-404.

Gokhale, R., & Mathirajan, M. (2012). Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *The International Journal of Advanced Manufacturing Technology*, 60 (912), 1099-1110.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, Massachusetts: Addison-Wesley.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *In Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, Hillsdale, NJ, 41-49.

Goldberg, D. E., & Smith, R. E. (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. *In Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, Hillsdale, NJ, 59-68.

Gonçalves, J. F., & Resende, M. G. C. (2012). A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers & Operations Research*, 39 (2), 179-190.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy, K. A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, 287–326.

Granmo, O. C., Oommen, B. J., Myrer, S., & Olsen, M. G. (2006). Determining optimal polling frequency using a learning automata-based solution to the fractional knapsack problem. *In Proceedings of IEEE Conference on Cybernetics and Intelligent Systems*, Bangkok, 1-7.

Gray, A. E, Seidmann, A., & Stecke, K.E. (1993). A synthesis of decision models for tool management in automated manufacturing. *Management Science*, 39 (5) 549-567.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. *In Proceedings of Parallel Problem Solving from Nature*, Amsterdam, 137-144.

Hadad, B., & Eick, C. (1997). Supporting polyploidy in genetic algorithms using dominance vectors. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, & R. Eberhart, (Eds.). *Evolutionary Programming VI* (223-234). Berlin Heidelberg: Springer.

Hartman, J. C, & Perry, T. C. (2006). Approximating the solution of a dynamic, stochastic multiple knapsack problem. *Control and Cybernetics*, 35 (3), 535-550.

Hashemi, A. B., & Meybodi, M. R. (2009a). Cellular PSO: A PSO for dynamic environments. In Z. Cai, Z. Li, Z. Kang, & Y. Liu, (Eds.). *Advances in Computation and Intelligence* (422-433). Berlin Heidelberg: Springer.

Hashemi, A. B., & Meybodi, M. R. (2009b). A multi-role cellular PSO for dynamic environments. *In Proceedings of the Computer Conference (CSICC)*, Tehran, 412-417.

He, D., He, C., Jiang, L.-G., Zhu, H.-W., & Hu, G.-R. (2001). Chaotic characteristics of a one-dimensional iterative map with infinite collapses. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48 (7), 900-906.

He, Y., Zhang, X., Li, W., Li, X., Wu, W., & Gao, S. (2016). Algorithms for randomized time-varying knapsack problems. *Journal of Combinatorial Optimization*, 31, 95-117.

He, Y.-Y., Zhou, J.-Z., Xiang, X.-Q., Chen, H., & Qin, H. (2009). Comparison of different chaotic maps in particle swarm optimization algorithm for long-term cascaded hydroelectric system scheduling. *Chaos, Solitons & Fractals*, 42 (5), 3169-3176.

Hertz, A., Laporte, G., Mittaz, M., & Stecke, K. E. (1998). Heuristics for minimizing tool switches when scheduling part types on flexible machine. *IIE Transactions*, 30 (8), 689-694.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Michigan Ann Arbor: University of Michigan Press.

Hu, X., & Eberhart, R. C. (2002). Adaptive particle swarm optimization: detection and response to dynamic systems. *In Proceedings of congress on the Evolutionary Computation Congress (CEC),* Honolulu, HI, 1666-1670.

Huang, C. J., & Liao, L. M. (2012). An artificial immune based algorithm for parallel-machine scheduling with preference of machines. *In Proceedings of IEEE*

International Conference on Industrial Engineering and Engineering Management (IEEM), Hong Kong, 469-473.

Janson, S., & Middendorf, M. (2004). A hierarchical particle swarm optimizer for dynamic optimization problems. In G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, C. G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. D. Smith, & G. Squillero, (Eds.). *Applications of Evolutionary Computing* (513-524). Berlin Heidelberg: Springer.

Jin, Y., & Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9 (3), 303-317.

Jordehi, A. R. (2014). Particle swarm optimisation for dynamic optimisation problems: a review. *Neural Computing and Applications*, 25 (7), 1507-1516.

Kamosi, M., Hashemi, A. B., & Meybodi, M. R. (2010a). A hibernating multi-swarm optimization algorithm for dynamic environments. *In Proceedings of Second World Congress on Nature and Biologically Inspired Computing (NaBIC),* Fukouka, 363-369.

Kamosi, M., Hashemi, A. B., & Meybodi, M. R. (2010b). A new particle swarm optimization algorithm for dynamic environments. In B. K. Panigrahi, S. Das, P. N. Suganthan, & S. S. Dash, (Eds.). *Swarm, Evolutionary, and Memetic Computing* (129-138). Berlin Heidelberg: Springer.

Karaman, A., & Uyar, A. S. (2004). A novel change severity detection mechanism for the dynamic 0/1 knapsack problem. *In Proceedings of the 10th International Conference on Soft Computing*.

Karaman, A., Uyar, Ş., & Eryiğit, G. (2005). The memory indexing evolutionary algorithm for dynamic environments. In F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith, & G. Squillero, (Eds.). *Applications of Evolutionary Computing* (563-573). Berlin Heidelberg: Springer.

Karimi, J., Nobahari, H., & Pourtakdoust, S. H. (2012). A new hybrid approach for dynamic continuous optimization problems. *Applied Soft Computing*, 12 (3), 1158-1167.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

Kennedy, J., & Mendes, R. (2002). Population structure and particle swarm performance. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Honolulu, HI, 1671-1676.

Keung, K. W., Ip, W. H, & Lee, T. C. (2001). The solution of a multi-objective tool selection model using the GA approach. *The International Journal of Advanced Manufacturing Technology*, 18 (11), 771-777.

Khadidja, Y., Fatima, D., & Fayçal, K. M. (2012). Hybrid algorithm based on HBMO and GRASP for real-time task scheduling problem resolution. *IJCSI International Journal of Computer Science Issues*, 9 (4), 1694-0814.

Kim, C. O., & Shin, H. J. (2003). Scheduling jobs on parallel machines: a restricted tabu search approach. *International Journal of Advanced Manufacturing Technology*, 22 (3), 278–287.

Kirkpatrick, S., Gelatt, C. D, & Vecchi, M. P. (1983). Optimisation by simulated annealing. *Science*, 220, 671-680.

Kleywegt, A. J., & Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations Research*, 46 (1), 17-35.

Kleywegt, A. J., & Papastavrou, J. D. (2001). The dynamic and stochastic knapsack problem with random sized items. *Operations Research*, 49 (1), 26-41.

Klinkmeijer, L. Z., de Jong, E., & Wiering, M. (2006). A serial population genetic algorithm for dynamic optimization problems. *In Proceedings of the 15th Belgian-Dutch Conference on Machine Learning*, 41-48.

Kordestani, J., Rezvanian, A., & Meybodi, M. (2014). CDEPSO: a bi-population hybrid approach for dynamic optimization problems. *Applied Intelligence*, 40 (4), 682-694.

Krishnakumar, K. (1990). Micro-genetic algorithms for stationary and non-stationary function optimization. *In Proceedings of SPIE 1196 International Conference on Intelligent Control and Adaptive Systems*, Philadelphia, USA, 289-296.

Kumaraguruparan, N., Sivaramakrishnan, H., & Sapatnekar, S. S. (2012). Residential task scheduling under dynamic pricing using the multiple knapsack method. *In Proceedings of IEEE Conference on Innovative Smart Grid Technologies (ISGT)*, Washington, 1-6.

Laporte, G., Salazar-Gonzalez, J. J., & Semet. F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 36 (1) 37-45.

Lee, J. K., Fox, M. S., & Watkins, P. R. (1993). Scheduling expert systems and their performances. *Expert Systems with Applications*, 6 (3), 217.

Lee, K., Leung, J. Y. T., & Pinedo, M. L. (2013). Makespan minimization in online scheduling with machine eligibility. *Annals of Operations Research*, 204 (1), 189-222.

Lee, Y. H., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29, 45–52.

Lepagnot, J., Nakib, A., Oulhadj, H., & Siarry, P. (2013). A multiple local search algorithm for continuous dynamic optimization. *Journal of Heuristics*, 19 (1), 35-76.

Levitin, G., & Rubinovitz, J. (1995). Algorithm for tool placement in an automatic tool change magazine. *International Journal of Production Research*, 33 (2), 351-360.

Li, C., & Yang, S. (2008). Fast multi-swarm optimization for dynamic optimization problems. *In Proceedings of the IEEE Conference on Natural Computation (ICNC)*, Jinan, 624-628.

Li, C., & Yang, S. (2009). A clustering particle swarm optimizer for dynamic optimization. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Trondheim, 439-446.

Li, C., & Yang, S. (2012). A general framework of multipopulation methods with clustering in undetectable dynamic environments. *IEEE Transactions on Evolutionary Computation*, 16 (4), 556-577.

Li, C., Yang, S., Nguyen, T. T., Yu, E. L., Yao, X., Jin, Y., Beyer, H. C, & Suganthan, P. N. (2008). *Benchmark generator for CEC'2009 competition on dynamic optimization*. Technical Report No: *Na*, University of Leicester, University of Birmingham, Nanyang Technological University.

Li, C., Yang, S., & Yang, M. (2014). An adaptive multi-swarm optimizer for dynamic optimization problems. *Evolutionary Computation*, 22 (4), 559-594.

Li, W. (2011). A parallel multi-start search algorithm for dynamic traveling salesman problem. In P. M. Pardalos, & S. Rebennack, (Eds.). *Experimental Algorithms* (65-75). Berlin Heidelberg: Springer.

Li, X. (2004). Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In K. Deb, (Ed.). *Genetic and Evolutionary Computation – GECCO 2004* (105-116). Berlin Heidelberg: Springer.

Li, X., Branke, J., & Blackwell, T. (2006). Particle swarm with speciation and adaptation in a dynamic environment. *In Proceeding of Genetic and Evolutionary Computation Conference (GECCO)*, Washington, USA, 51-58.

Liekens, A. M. L. (2005). *Evolution of finite populations in dynamic environments*. Phd Thesis, Technische Universitat, Eindhoven.

Lin, F. T., Kao, C. Y., & Hsu, C. C. (1993). Applying the genetic approach to simulated annealing in solving some NP-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23 (6), 1752-1767.

Lin, S. W. (2013). Solving the team orienteering problem using effective multi-start simulated annealing. *Applied Soft Computing*, 13 (2), 1064-1073.

Lin, S. W., Ying, K. C., Lu, C. C., & Gupta, J. N. (2011). Applying multi-start simulated annealing to schedule a flowline manufacturing cell with sequence dependent family setup times. *International Journal of Production Economics*, 130 (2), 246-254

Lin, Y. K., & Hsieh, F. Y. (2014). Unrelated parallel machine scheduling with setup times and ready times. *International Journal of Production Research*, 52 (4), 1200-1214.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20 (2), 130-141.

Lung, R. I., & Dumitrescu, D. (2007). A new collaborative evolutionary-swarm optimization technique. *In Proceedings of Companion on Genetic and Evolutionary Computation*, London, United Kingdom, 2817-2820.

Lung, R., & Dumitrescu, D. (2010). Evolutionary swarm cooperative optimization in dynamic environments. *Natural Computing*, 9 (1), 83-94.

Matzliach, B. (1998). The online tool switching problem with non-uniform tool size. *International Journal of Production Research*, 36 (12), 3407-3420.

Mavrovouniotis, M. (2013). *Ant colony optimization in stationary and dynamic environments*. Phd Thesis, University of Leicester, Leicester.

May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, 261 (5560), 459-467.

Mendes, R., & Mohais, A. S. (2005). DynDE: a differential evolution for dynamic optimization problems. *In Proceedings of IEEE Congress on Evolutionary Computation*, Edinburgh, 2808-2815.

Mingjun, J., & Huanwen, T. (2004). Application of chaos in simulated annealing. *Chaos, Solitons & Fractals*, 21 (4), 933-941.

Mitchell, M., Forrest, S., & Holland, J. H. (1992). The royal road for genetic algorithms: fitness landscapes and GA performance. *In Proceedings of the First European Conference on Artificial Life*, Cambridge: The MIT Press, 245-254.

Montemanni, R., Gambardella, L. M., Rizzoli, A., & Donati, A. (2003). A new algorithm for a dynamic vehicle routing problem based on ant colony system. *In Second International Workshop on Freight Transportation and Logistics*, 27-30.

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10 (4), 327-343.

Moraga, R. J. (2002). *Meta-RaPS: an effective solution approach for combinatorial problems*. PhdThesis, University of Central Florida, Orlando, FL.

Moraga, R. J., DePuy, G. W., & Whitehouse, G. E. (2005). Meta-RaPS approach for the 0-1 multidimensional knapsack problem. *Computers & Industrial Engineering*, 48, 83–96.

Morales-Enciso, S., & Branke, J. (2015). Tracking global optima in dynamic environments with efficient global optimization. *European Journal of Operational Research*, 242 (3), 744-755.

Mori, N., Kita, H., & Nishikawa, Y. (1996). Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel, (Eds.). *Parallel Problem Solving from Nature—PPSN IV* (513-522). Berlin Heidelberg: Springer.

Morrison, R. W. (2003). Performance measurement in dynamic environments. *In GECCO workshop on evolutionary algorithms for dynamic optimization problems*, Chicago, Illinois, 5-8.

Morrison, R. W. (2004). *Designing evolutionary algorithms for dynamic environments*. Berlin: Springer Verlag.

Morrison, R. W., & De Jong, K. A. (1999). A test problem generator for non-stationary environments. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*,Washington.

Nabizadeh, S., Rezvanian, A., & Meybodi, M. R. (2012a). A multi-swarm cellular PSO based on clonal selection algorithm in dynamic environments. *In Proceedings of the International Conference on Informatics, Electronics & Vision (ICIEV)*, Bangladesh, 482-486.

Nabizadeh, S., Rezvanian, A., & Meybodi, M. R. (2012b). Tracking extrema in dynamic environment using multi-swarm cellular PSO with local search. *International Journal of Electronics & Informatics*, 1 (1), 29-37.

Nanayakkara, T., Watanabe, K., & Izumi, K. (1999). Evolving in dynamic environments through adaptive chaotic mutation. *In Proceedings of the third International Symposium on Artificial Life and Robotics*, Beppu, Japan, 520-523.

Nasiri, B., & Meybodi, M. (2012). Speciation based firefly algorithm for optimization in dynamic environments. *International Journal of Artificial Intelligence*, 8 (12), 118-132.

Nguyen, T. T. (2010). *Continuous dynamic optimisation using evolutionary algorithms*. Phd Thesis, 2010, The University of Birmingham, Birmingham.

Nguyen, T. T., Yang, S., & Branke, J. (2012). Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6, 1-24.

Noroozi, V., Hashemi, A., & Meybodi, M. (2011). CellularDE: a cellular based differential evolution for dynamic optimization problems. In A. Dobnikar, U. Lotrič, & B. Šter, (Eds.). *Adaptive and Natural Computing Algorithms* (340-349). Berlin Heidelberg: Springer.

Oppacher, F., & Wineberg, M. (1999). The shifting balance genetic algorithm: Improving the GA in a dynamic environment. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Orlando, Florida, 504-510.

Ott, E. (2002). *Chaos in dynamical systems*. Cambridge, UK: Cambridge university press.

Ovacik, I. M., & Uzsoy, R. (1995a). *Benchmark data*. Retrieved *Na*, from http://cobweb.ecn.purdue.edu/~uzsoy2/Problems/parallel/parameters.html.

Ovacik, I. M., & Uzsoy, R. (1995b). Rolling horizon procedures for dynamic parallel machine scheduling with sequence-dependent setup times. *International Journal of Production Research*, 33 (11), 3173-3192.

Ozsoydan, F. B., & Baykasoglu, A. (2015). A multi-population firefly algorithm for dynamic optimization problems. *In Proceedings of IEEE conference on Evolving and Adaptive Intelligent Systems (EAIS)*, Douai, 1-7.

Parrott, D., & Li, X. (2004). A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. *In Proceedings of the Congress on Evolutionary Computation (CEC)*, 98-103.

Parrott, D., & Li, X. (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation*, 10 (4), 440-458.

Pelta, D., Cruz, C., & Verdegay, J. L. (2009). Simple control rules in a cooperative system for dynamic optimisation problems. *International Journal of General Systems*, 38 (7), 701-717.

Perry, T. C., & Hartman, J. C. (2009). An approximate dynamic programming approach to solving a dynamic, stochastic multiple knapsack problem. *International Transactions in Operational Research*, 16 (3), 347-359.

Petrowski, A. (1996). A clearing procedure as a niching method for genetic algorithms. *In Proceedings of IEEE International Conference on Evolutionary Computation (CEC)*, Nogaya, Japan, 798-803.

Pfund, M., Fowler, J. W., & Gupta, J. N. D. (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. *Journal of the Chinese Institute of Industrial Engineers*, 21 (3), 230–241.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

Qin, A. K., Huang, V. L., & Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13 (2), 398-417.

Rand, W., & Riolo, R. (2005). Shaky ladders, hyperplane-defined functions and genetic algorithms: systematic controlled observation in dynamic environments. In Franz Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. G. Smith, & G. Squillero, (Eds.). *Applications of Evolutionary Computing* (600-609). Berlin Heidelberg: Springer.

Randall, M. (2005). A dynamic optimisation approach for ant colony optimisation using the multiple knapsack problem. *In Proceedings of the Second Australian Conference on Artificial Life*, Sydney, 1-14.

Rezazadeh, I., Meybodi, M., & Naebi, A. (2011). Adaptive particle swarm optimization algorithm for dynamic environments. In Y. Tan, Y. Shi, Y. Chai, & G. Wang, (Eds.). *Advances in Swarm Intelligence* (120-129). Berlin Heidelberg: Springer.

Richter, H. (2010). Memory design for constrained dynamic optimization problems. In C. D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcazar, C.-K. Goh, J. J. Merelo, F. Neri, M. Preuß, J. Togelius, & G. Yannakakis, (Eds.). *Applications of Evolutionary Computation* (552-561). Berlin Heidelberg: Springer.

Richter, H., & Yang, S. (2008). Memory based on abstraction for dynamic fitness functions. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, R. Drechsler, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, J. McCormack, M. O'Neill, J. Romero, F. Rothlauf, G. Squillero, A. Ş. Uyar, & S. Yang, (Eds.). *Applications of Evolutionary Computing* (596-605). Berlin Heidelberg: Springer.

Richter, H., & Yang, S. (2009). Learning behavior in abstract memory schemes for dynamic optimization problems. *Soft Computing*, 13 (12), 1163-1173.

Rohlfshagen, P., & Bullinaria, J. (2006). Alternative splicing in evolutionary computation: Adaptation in dynamic environments. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, Vancouver, BC, 2277-2284.

Rohlfshagen, P., & Yao, X. (2008). Attributes of dynamic combinatorial optimization. In X. Li, M. Kirley, M. Zhang, D. Green, V. Ciessielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, & Y. Shi, (Eds.). *Simulated Evolution and Learning* (442-451). Berlin Heidelberg: Springer.

Rohlfshagen, P., & Yao, X. (2009). The dynamic knapsack problem revisited: a new benchmark problem for dynamic combinatorial optimisation. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, & P. Machado, (Eds.). *Applications of Evolutionary Computing (*745-754). Berlin Heidelberg: Springer.

Rohlfshagen P, & Yao X. (2011). Dynamic combinatorial optimisation problems: an analysis of the subset sum problem. *Soft Computing*, 15 (9), 1723–34.

Rossi, C., Barrientos, A., & Cerro, J. (2007). Two adaptive mutation operators for optima tracking in dynamic optimization problems with evolution strategies. *In*

Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO, London, 697-704. Berlin Heidelberg: Springer.

Ruiz-Torres, A. J., Paletta, G., & Pérez, E. (2013). Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 40 (8), 2051-2061.

Ryan, C. (1997). Diploidy without dominance. *In Proceedings of the Third Nordic Workshop on Genetic Algorithms*, Helsinki, Finland, (63-70).

Salonen, K., Raduly-Baka, C., & Nevalainen, O. S. (2006). A note on the tool switching problem of a flexible machine. *Computers & Industrial Engineering*, 50 (4), 458-465.

Schönemann, L. (2004). The impact of population sizes and diversity on the adaptability of evolution strategies in dynamic environments. *In Proceedings of IEEE Congress on Evolutionary Computation*, 1270-1277.

Schönemann, L. (2007). Evolution strategies in dynamic environments. In *Studies in computational intelligence* (51-77), New York: Springer.

Sepas-Moghaddam, A., Yazdani, D., Arabshahi, A., & Dehshibi, M. M. (2012). A novel hybrid algorithm for optimization in multimodal dynamic environments. *In Proceedings of the 12th International Conference on Hybrid Intelligent Systems (HIS)*, Pune, 143-148.

Sharifi, A., Noroozi, V., Bashiri, M., Hashemi, A. B., & Meybodi, M. R. (2012). Two phased cellular PSO: a new collaborative cellular algorithm for optimization in dynamic environments. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Australia, pp. 1-8.

Shi, Y., & Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. *In Proceedings of IEEE Congress on Evolutionary Computation*, Seoul, 101-106.

Shlesinger, M. F., Zaslavsky, G. M., & Frisch, U. (1995). Lévy flights and related topics in physics. In M. F. Shlesinger, G. M. Zaslavsky, & U. Frisch, (Eds.). *Lévy Flights and Related Topics in Physics*. Berlin Heidelberg: Springer.

Simões, A., & Costa, E. (2001). Using biological inspiration to deal with dynamic environments. *In the Proceedings of the Seventh International Conference on Soft Computing (MENDEL)*, Brno, Czech Republic.

Simões, A., & Costa, E. (2002). Using genetic algorithms to deal with dynamic environments: a comparative study of several approaches based on promoting diversity. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO),* New York.

Simões, A., & Costa, E. (2007a). Improving memory's usage in evolutionary algorithms for changing environments. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, Singapore, 276-283.

Simões, A., & Costa, E. (2007b). Variable-size memory evolutionary algorithm to deal with dynamic environments. In M. Giacobini, (Ed.). *In Applications of Evolutionary Computing* (617-626). Berlin Heidelberg: Springer.

Simões, A., & Costa, E. (2008). Evolutionary algorithms for dynamic environments: prediction using linear regression and markov chains. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, & C. Poloni, (Eds.). *Parallel Problem Solving from Nature – PPSN X* (306-315). Berlin Heidelberg: Springer.

Simões, A., & Costa, E. (2009a). Improving prediction in evolutionary algorithms for dynamic environments. *In Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, NY, 875-882.

Simões, A., & Costa, E. (2009b). Prediction in evolutionary algorithms for dynamic environments using markov chains and nonlinear regression. *In Proceedings of Genetic and Evolutionary Computation Conference*, ACM, NY, 883-890.

Simões, A., & Costa, E. (2009c). The influence of population and memory sizes on the evolutionary algorithm's performance for dynamic environments. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, & P. Machado, (Eds.). *In Applications of Evolutionary Computing* (705-714). Berlin Heidelberg: Springer.

Sinriech, D., Rubinovitz, J., Milo, D., & Nakbily, G. (2001). Sequencing, scheduling and tooling single-stage multifunctional machines in a small batch environment. *IIE Transactions*, 33 (10), 897-911.

Smith, J. E., & Vavak, F. (1999). Replacement strategies in steady state genetic algorithms: dynamic environments. *Journal of Computing and Information Technology*, 7, 49-60.

Stanhope, S. A., & Daida, J. M. (1998). Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In V. W. Porto, N. Saravanan, D. Waagen, & A. E. Eiben, (Eds.). *Evolutionary Programming VII* (693-702). Berlin: Springer Heidelberg.

Stanhope, S. A., & Daida, J. M. (1999). (1+ 1) genetic algorithm fitness dynamics in a changing environment. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*,Washington.

Stecke, K. E. (1983). Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, 29 (3), 273-288.

Storn, R., & Price, K. (1995). *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical Report No: TR-95-012, ICSI, CA: International Computer Science Institute, Berkeley.

Tang, C. S., & Denardo, E. V. (1988). Models arising from flexible manufacturing machine, minimization of the number of tool switches. *Operations Research*, 36 (5), 767-784.

Taylor, D., Weber, B., & Bojduj, B. (2006). A tabu search framework for dynamic combinatorial optimization problems. *In Proceedings of Integrated Design and Process Technology (IDPT)*, USA, 663–668.

Thomsen, R. (2004). Multimodal optimization using crowding-based differential evolution. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, 1382-1389.

Tinós, R., & Yang, S. (2007). A self-organizing random immigrants genetic algorithm for dynamic optimization problems. *Genetic Programming and Evolvable Machines*, 8 (3), 255-286.

Toledo, C. F. M., de Oliveira, R. R. R., & Morelato França, P. (2013). A hybrid multi-population genetic algorithm applied to solve the multi-level capacitated lot sizing problem with backlogging. *Computers & Operations Research*, 40 (4), 910-919.

Trojanowski, K., & Michalewicz, Z. (1999). Searching for optima in non-stationary environments. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Washington.

Tsutsui, S., Fujimoto, Y., & Ghosh, A. (1997). Forking genetic algorithms: GAs with search space division schemes. *Evolutionary Computation*, 5 (1), 61-80.

Turkcan, A., Akturk, M. S., & Storer, R. H. (2007). Due date and cost-based FMS loading, scheduling and tool management. *International Journal of Production Research*, 45 (5), 1183-1213.

Ullman, J. D. (1975). NP-complete scheduling problem. *Journal of Computing System Science*, 10, 384–393.

Ursem, R. K. (1999). Multinational evolutionary algorithms. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Washington, DC, 1633-1640.

Ursem, R. K. (2000). Multinational GAs: multimodal optimization techniques in dynamic environments. *In Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, 19-26.

Uyar, A. Ş. (2007). Experimental comparison of replacement strategies in steady state genetic algorithms for the dynamic MKP. In M. Giacobini, (Ed.). *In Applications of Evolutionary Computing* (647-656). Berlin Heidelberg: Springer.

Uyar, A. Ş., & Harmanci, A. E. (2005). A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. *Soft Computing*, 9 (11), 803-814.

Uyar, Ş., & Uyar, H. T. (2009). A critical look at dynamic multi-dimensional knapsack problem generation. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, & P. Machado, (Eds.). *Applications of Evolutionary Computing* (762-767). Berlin Heidelberg: Springer.

Ünal, A. N., & Kayakutlu, G. (2016). A partheno-genetic algorithm for dynamic 0-1 multidimensional knapsack problem. *RAIRO-Operations Research*, 50 (1), 47-66.

van Hentenryck, P., & Vergados, Y. (2007). Population-based simulated annealing for traveling tournaments. *In Proceedings of the National Conference on Artificial Intelligence*, 267-272.

van Hemert, J., van Hoyweghen, C., Lukschandl, E., & Verbeeck, K. (2001). A futurist approach to dynamic environments. *In GECCO EvoDOP Workshop*, 35-38.

van Hop, N. (2005). The tool-switching problem with magazine capacity and tool size constraints. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 35 (5), 617-628.

Vavak, F., Jukes, K., & Fogarty, T. C. (1998). Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes. *Genetic Programming*, 22-25.

Wang, H., Yang, S., Ip, W. H., & Wang, D. (2009). Adaptive primal–dual genetic algorithms in dynamic environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39 (6), 1348-1361.

Watson, J. P. (1999). A performance assessment of modern niching methods for parameter optimization problems. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Orlando, Florida.

Weicker, K. (2002). Performance measures for dynamic environments. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, & J.-L. Fernández-Villacañas, (Eds.). *Parallel Problem Solving from Nature—PPSN VII* (64-73). Berlin Heidelberg: Springer.

Weicker, K. (2003a). *Evolutionary algorithms and dynamic optimization problems*. Berlin: Der Andere Verlag.

Weicker, K. (2003b). *Evolutionary algorithms and dynamic optimization problems*. Phd Thesis, Universitat Stuttgart: Stuttgart.

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. In G. J. E.Rawlins, (Ed.). *Foundations of Genetic Algorithms* (221-241). San Mateo, California: Morgan Kaufmann Publishers.

Wilke, C. (1999). *Evolutionary dynamics in time dependent environments*. Phd Thesis, Universitat Bochum: Shaker.

Wilson, J. M. (1987). Formulation and solution of a set of sequencing problems for flexible manufacturing systems. *Journal of Engineering Manufacture*, 201 (4), 247-249.

Wineberg, M., & Chen, J. (2004). The shifting balance genetic algorithm as more than just another island model GA. In K. Deb, (Ed.). *Genetic and Evolutionary Computation – GECCO 2004* (318-329). Berlin Heidelberg: Springer.

Woldesenbet, Y. G., & Yen, G. G. (2009). Dynamic evolutionary algorithm with variable relocation. *IEEE Transactions on Evolutionary Computation*, 13 (3), 500-513.

Wu, Y., Wang, Y., & Liu, X. (2010). Multi-population based univariate marginal distribution algorithm for dynamic optimization problems. *Journal of Intelligent & Robotic Systems*, 59 (2), 127-144.

Xiao, J., Yang, H., Zhang, C., Zheng, L., & Gupta, J. N. (2015). A hybrid lagrangian-simulated annealing-based heuristic for the parallel-machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 63, 72-82.

Xiao, L., & Zuo, X. (2012). Multi-DEPSO: A DE and PSO based hybrid algorithm in dynamic environments. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Brisbane, QLD, 1-7.

Yan, Y., Wang, D., Wang, H., & Wang, D. (2008). Multi-agent based evolutionary algorithm for dynamic knapsack problem. *In Proceedings of Control and Decision Conference (CCDC)*, Yantai, Shandong, 4215-4220.

Yang, S. (2003). Non-stationary problem optimization using the primal dual genetic algorithm. *In Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Newport Beach, California, 2246–2253.

Yang, S. (2005). Memory-based immigrants for genetic algorithms in dynamic environments. *In Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, NY, USA, 1115-1122.

Yang, S. (2006a). Associative memory scheme for genetic algorithms in dynamic environments. In F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R.

Drechsler, E. Lutton, P. Machado, J. Moore, J. Romero, G. Smith, G. Squillero, & H. Takagi, (Eds.). *Applications of Evolutionary Computing* (788-799). Berlin Heidelberg: Springer.

Yang, S. (2006b). On the design of diploid genetic algorithms for problem optimization in dynamic environments. *In Proceedings of the IEEE Congress on Evolutionary Computation, (CEC)*, Vancouver, BC, 1362-1369.

Yang, S. (2007). Genetic algorithms with elitism-based immigrants for changing optimization problems. In M. Giacobini, (Ed.). *Applications of Evolutionary Computing* (627-636). Berlin Heidelberg: Springer.

Yang, S. (2008). Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, 16 (3), 385-416

Yang, S., Cheng, H., & Wang, F. (2010). Genetic algorithms with immigrants and memory schemes for dynamic shortest path routing problems in mobile Ad Hoc Networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40 (1), 52-63.

Yang, S., Jiang, Y., & Nguyen, T. T. (2013). Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, 24 (4), 451–480.

Yang, S., & Li, C. (2010). A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 14 (6), 959-974.

Yang, S., & Richter, H. (2009). Hyper-learning for population-based incremental learning in dynamic environments. *In IEEE Congress on Evolutionary Computation (CEC)*, Trondheim, 682-689.

Yang, S., & Tinós, R. (2008). Hyper-selection in dynamic environments. *In Proceedings of IEEE World Congress on Computational Intelligence, Evolutionary Computation*, Hong Kong, 3185-3192.

Yang, S., & Xin, Y. (2008). Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*, 12 (5), 542-561.

Yang, X.-S. (2008). *Nature-inspired metaheuristic algorithms*. Frome, Somerset: Luniver Press.

Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. In O. Watanabe, & T. Zeugmann, (Eds.). *Stochastic Algorithms: Foundations and Applications* (169-178). Berlin Heidelberg: Springer.

Yang, X.-S. (2010). Firefly algorithm, lévy flights and global optimization. In M. Bramer, R. Ellis, & M. Petridis, (Eds.). *Research and Development in Intelligent Systems XXVI* (209-218). London: Springer.

Yang, S., & Yao, X. (2003). Dual population-based incremental learning for problem optimization in dynamic environments. *In Proceedings of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems*, 49-56.

Yang, S., & Yao, X. (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9, 815-834.

Yazdani, D., Akbarzadeh-Totonchi, M. R., Nasiri, B., & Meybodi, M. R. (2012). A new artificial fish swarm algorithm for dynamic optimization problems. *In Proceeding of IEEE Congress on Evolutionary Computation (CEC)*, Brisbane, QLD, 1-8.

Yazdani, D., Nasiri, B., Azizi, R., Sepas-Moghaddam, A., & Meybodi, M. R. (2013). Optimization in dynamic environments utilizing a novel method based on particle swarm optimization. *International Journal of Artificial Intelligence*, 11 (A13), 170-192.

Yazdani, D., Nasiri, B., Sepas-Moghaddam, A., & Meybodi, M. R. (2013). A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. *Applied Soft Computing*, 13 (4), 2144-2158.

Yazdani, D., Nasiri, B., Sepas-Moghaddam, A., Meybodi, M., & Akbarzadeh-Totonchi, M. (2014). mNAFSA: A novel approach for optimization in dynamic environments with global changes. *Swarm and Evolutionary Computation*, 18, 38-53.

Yilmaz Eroglu, D., Ozmutlu, H. C., & Ozmutlu, S. (2014). Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research*, 52 (19), 5841-5856.

Ying, K. C., & Cheng, H. M. (2010). Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, 37 (4), 2848-2852.

Yu, E. L., & Suganthan, P. N. (2009). Evolutionary programming with ensemble of explicit memories for dynamic optimization. *In Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, Trondheim, 431-438.

APPENDICES

A1 Data Set of Static Problem Introduced in Chapter 6.

number of operations	*ID of the problem*	*ATC indexing LB*	*LS*	*number of operations*	*ID of the problem*	*ATC indexing LB*	*LS*
50	1	410	10	30	6	390	15
50	2	516	12	30	7	338	13
50	3	470	10	30	8	324	12
50	4	572	13	30	9	240	10
50	5	690	15	30	10	280	10
50	6	552	12	20	1	154	11
50	7	506	11	20	2	270	15
50	8	611	13	20	3	216	12
50	9	672	14	20	4	255	15
50	10	602	14	20	5	209	11
30	1	312	12	20	6	190	10
30	2	297	11	20	7	224	14
30	3	405	15	20	8	247	13
30	4	324	12	20	9	198	11
30	5	275	11	20	10	180	12

- There exist 16 index positions on the ATC and each tool has three available copies.
- Process plans and required tool types for each part are presented in Appendix 2 and in Baykasoğlu & Ozsoydan (2016) as well.

A2 Process Plans for Static Problem Introduced in Chapter 6

The data is adopted from Baykasoğlu & Ozsoydan (2016).

Benchmark Suite-1 (small scaled instances). There are 8 tool types each with 2 copies and maximum capacity of turret is 12.

	Oper. No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	*LB*
#1	Tools	4	5	5	4	3	3	5	4	8	4	2	8	2	2	8	8	2	4	4	1	*14*
#2	Tools	5	1	3	3	8	6	1	3	2	6	2	7	3	4	8	1	3	2	3	6	*18*
#3	Tools	6	7	3	4	6	4	5	8	1	5	6	7	3	2	3	3	2	7	5	4	*18*
#4	Tools	7	4	4	7	6	4	1	8	4	8	2	8	5	7	7	8	2	4	5	8	*17*
#5	Tools	1	5	2	5	7	8	5	6	2	7	6	2	8	6	4	1	8	4	7	5	*19*
#6	Tools	8	2	1	4	3	2	1	8	2	1	2	1	7	4	1	8	2	4	2	8	*19*
#7	Tools	5	3	7	2	6	2	5	4	8	3	7	3	5	6	6	2	1	2	2	2	*16*
#8	Tools	5	6	7	6	2	7	1	6	2	8	3	4	7	8	4	3	7	6	1	2	*19*
#9	Tools	2	6	1	6	8	5	7	1	7	1	4	1	6	7	3	6	7	3	3	4	*18*
#10	Tools	4	7	7	6	2	1	6	3	5	2	2	3	8	8	3	2	1	1	5	7	*15*

Benchmark Suite-2 (medium scaled instances). There are 8 tool types each with 2 copies and maximum capacity of turret is 12.

	Oper. No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	*LB*
#1	Tools	3	4	6	8	7	1	3	5	3	1	7	5	2	3	5	1	7	7	8	7	2	5	2	6	1	7	7	6	6	3	*26*
#2	Tools	2	7	1	5	5	6	4	6	1	8	5	7	8	5	4	8	5	7	8	6	4	4	1	5	8	3	5	8	5	2	*27*
#3	Tools	6	7	8	6	2	5	7	7	3	5	2	3	6	5	5	4	8	4	5	8	4	6	2	4	5	4	6	5	6	3	*27*
#4	Tools	7	5	4	8	4	5	6	5	4	1	3	4	3	2	6	5	7	8	6	2	2	2	4	2	7	2	6	2	6	4	*27*
#5	Tools	6	1	4	7	6	3	1	4	4	6	2	3	7	3	8	6	8	6	6	3	2	4	3	3	4	4	1	5	3	4	*25*
#6	Tools	7	8	3	7	2	1	6	4	7	7	6	5	8	8	6	3	6	7	7	6	3	1	4	3	4	7	5	1	2	4	*26*
#7	Tools	7	8	5	4	5	7	4	6	1	7	6	4	7	8	2	2	5	4	2	3	7	5	5	6	8	3	3	1	7	3	*26*
#8	Tools	7	4	3	3	1	7	4	7	4	5	7	6	8	4	8	2	4	6	3	4	7	2	6	7	7	5	8	1	5	1	*27*
#9	Tools	1	1	1	7	6	6	8	6	3	3	8	3	1	2	7	8	7	7	3	2	3	1	5	7	6	1	6	5	1	7	*24*
#10	Tools	8	6	2	3	2	3	5	1	3	5	8	3	8	3	2	4	1	7	7	1	4	8	6	2	4	1	6	4	3	7	*28*

Benchmark Suite-3 (large scaled instances). There are 12 tool types each with 2 copies and maximum capacity of turret is 16.

Oper. No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	*LB*
#1 Tools	8	8	11	2	3	5	8	1	9	1	1	11	4	4	9	6	6	11	1	10	2	12	4	3	3	12	11	3	8	5	4	4	10	12	2	5	2	6	2	7	9	3	12	12	1	10	2	4	4	9	*41*
#2 Tools	4	6	12	4	11	12	6	12	11	1	12	3	10	12	11	8	9	7	11	10	5	1	2	12	5	11	3	7	8	8	9	1	12	3	2	10	10	10	6	6	6	8	9	11	7	7	4	10	4	2	*43*
#3 Tools	8	7	8	10	8	12	7	8	12	10	6	8	4	12	9	7	4	10	8	10	11	5	7	3	7	6	12	11	2	9	2	5	3	6	12	9	9	10	5	5	6	12	6	2	11	5	8	6	10	1	*47*
#4 Tools	6	8	9	5	3	4	5	3	4	12	7	4	10	11	11	12	6	12	7	7	5	2	4	1	11	10	8	4	4	7	1	12	12	7	1	2	12	7	7	10	6	7	1	4	1	6	7	12	10	1	*44*
#5 Tools	4	12	11	5	6	1	11	7	2	6	5	1	8	8	5	8	12	8	4	4	11	12	3	6	7	7	10	4	6	12	6	12	9	10	8	1	10	1	12	5	9	1	5	8	7	1	8	12	1	5	*46*
#6 Tools	1	6	12	5	6	7	5	9	3	5	6	1	4	6	6	11	5	9	6	7	2	3	5	9	8	2	10	2	8	1	9	3	3	12	4	6	5	12	3	2	5	6	12	8	7	7	9	12	7	11	*46*
#7 Tools	7	10	8	2	8	8	3	6	7	1	11	7	1	11	2	10	9	9	6	4	5	8	9	8	1	2	11	2	10	2	5	6	3	6	7	7	2	6	3	12	9	8	12	10	8	7	1	2	6	1	*46*
#8 Tools	11	3	2	5	4	7	5	12	7	3	4	5	9	10	7	6	11	9	5	10	2	6	10	12	12	9	8	2	1	10	5	10	2	7	5	2	9	1	4	4	8	1	6	10	1	12	8	3	12	11	*47*
#9 Tools	6	9	7	10	2	5	3	12	12	3	6	2	3	4	3	7	10	9	1	2	8	10	4	1	10	6	7	8	4	7	11	5	2	9	4	6	11	9	6	12	11	12	5	2	8	12	7	6	2	1	*48*
#10 Tools	6	6	6	10	5	3	5	12	8	9	5	12	4	12	6	11	12	3	9	1	10	2	8	1	12	2	3	6	9	4	6	3	7	7	5	6	12	12	8	12	6	6	6	3	1	5	8	2	9	8	*43*