A GENETIC ALGORITHM FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM HAVING A SINGLE MACHINE WITH SEQUENCE DEPENDENT SETUP TIMES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SÜLEYMAN KAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF INDUSTRIAL ENGINEERING

FEBRUARY 2013

Approval of the thesis:

A GENETIC ALGORITHM FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM HAVING A SINGLE MACHINE WITH SEQUENCE DEPENDENT SETUP TIMES

submitted by **Süleyman KAYA** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan ÖZGEN
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Murat KÖKSALAN
Head of Department, **Industrial Engineering** _____

Assoc. Prof. Dr. Sedef MERAL
Supervisor, **Industrial Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Sinan KAYALIGİL
Industrial Engineering Dept., METU _____

Prof. Dr. Nur Evin ÖZDEMİREL
Industrial Engineering Dept., METU _____

Prof. Dr. Meral AZİZOĞLU
Industrial Engineering Dept., METU _____

Assoc. Prof. Dr. Sedef MERAL
Industrial Engineering Dept., METU _____

Assoc. Prof. Dr. Ferda Can ÇETİNKAYA
Industrial Engineering Dept., ÇANKAYA U. _____

Date : 01.02.2013

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Süleyman KAYA

Signature :

ABSTRACT

A GENETIC ALGORITHM FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM HAVING A SINGLE MACHINE WITH SEQUENCE DEPENDENT SETUP TIMES

Kaya, Süleyman
M.S., Department of Industrial Engineering
Supervisor: Assoc. Prof. Dr. Sedef Meral

February 2013, 65 Pages

The scheduling problem considered in this study is the integration of two different problems in the scheduling area. One of the problems is the resource constrained project scheduling problem with renewable resources, while the other one is the single machine scheduling problem with sequence dependent setup times. In real life, project scheduling problems are usually complicated and include various scheduling problems characteristics. The objective of the problem addressed is the minimization of the completion time of the project. A genetic algorithm and a MIP model are developed for the problem. The results of the genetic algorithm for small problem instances are compared with the results of the MIP model coded using the library of IBM ILOG CPLEX. The MIP model developed is the integration of the MIP model of the resource constrained project scheduling problem and the MIP model of the single machine scheduling with sequence dependent setup times. For big problem instances, results are compared with the results of hill-climbing-like search algorithm. Computer programs for the genetic algorithm, MIP model and the hill-climbing-like search algorithm are coded by Microsoft Visual C# .Net platform. The results obtained by the proposed genetic algorithm are always superior to the hill-climbing-like search algorithm's results.

Keywords: Single machine scheduling with sequence dependent setup times, Resource constrained project scheduling, Genetic algorithms, Search algorithms

ÖZ

SIRA BAĞIMLI KURULUM ZAMANLI TEK BİR MAKİNEYE SAHİP KAYNAK KISITLI PROJE ÇİZELGELEME PROBLEMİ İÇİN BİR GENETİK ALGORİTMA

Kaya, Süleyman
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Sedef Meral

Şubat 2013, 65 Sayfa

Bu çalışmada ele alınan çizelgeleme problemi çizelgeleme alanındaki iki farklı tipteki çizelgeleme probleminin birleşiminden oluşmaktadır. Bu problemlerden biri kaynak (yenilenebilir) kısıtlı proje çizelgeleme problemi, diğeri ise sıra bağımlı kurulum zamanlarına sahip tek makineli çizelgeleme problemidir. Gerçek hayatta proje çizelgeleme problemleri genellikle karmaşıktır ve çeşitli çizelgeleme problem karakteristiklerini içermektedir. Ele alınan problemdeki amaç projenin bitiş zamanını en aza indirmektir. Bu problem için bir genetik algoritma ve karışık tamsayılı doğrusal programlama modeli geliştirilmiştir. Küçük boyutlu problem örneklerinde genetik algoritmanın sonuçları IBM ILOG CPLEX kütüphanesi kullanılarak kodlanan karışık tamsayılı doğrusal programlama modelinin sonuçlarıyla karşılaştırılmıştır. Geliştirilen karışık tamsayılı programlama modeli, kaynak kısıtlı proje çizelgeleme probleminin karışık tamsayılı programlama modeli ile sıra bağımlı kurulum zamanlarına sahip tek makineli çizelgeleme probleminin karışık tamsayılı programlama modelinin birleşiminden oluşmaktadır. Büyük boyutlu problem örnekleri için genetik algoritmanın sonuçları, bir tepe-tırmanma benzeri arama algoritmasının sonuçlarıyla karşılaştırılmıştır. Genetik algoritma, karışık tamsayılı programlama modeli ve bir tepe-tırmanma benzeri arama algoritmasının bilgisayar programları Microsoft Visual C# .Net platform kullanılarak kodlanmıştır. Önerilen genetik algoritmanın verdiği sonuçların tepe-tırmanma benzeri algoritmanın verdiği sonuçlardan daima daha iyi olduğu gözlemlenir.

Anahtar Kelimeler: Sıra bağımlı kurulum zamanlarına sahip tek makineli çizelgeleme, Kaynak kısıtlı proje çizelgeleme, Genetik algoritmalar, Arama algoritmaları

To My Unique Father and Mother

ACKNOWLEDGEMENTS

I would like to thank my supervisors Assoc. Prof. Dr. Sedef Meral and Assoc. Prof. Dr. Canan Sepil for their continuous support, guidance, advice throughout this study.

I would like to express my deepest thanks to my parents, Yusuf Kaya and Ayşe Kaya. I am forever grateful to them for their understanding and encouragement.

Finally, I would like to thank to my sister, Habibe Kaya, who shared the burden of the study with me. She constantly believed in and motivated me, so that I complete this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The integrated problem addressed in this study is a resource constrained project scheduling problem with a special type of resource. The problem is adapted from a real life scheduling problem in a building construction project which is carried out in a construction area consisting of a number of sites. Similiar activities (tasks) are carried out at different sites, and the special resource, a crane, is used for some of the activities in this project. The crane can be used without a need for transport within the boundaries of a site. However, if an activity from a different site needs the crane, the crane should be taken to the site. Transportation of the crane requires a certain amount of time. When the problem is defined as such, it turns out to be an integrated problem which is a resource constrained project scheduling problem combined with the single machine scheduling problem having sequence dependent setup times.

In resource constrained project scheduling problems (RCPSP), there is a set of tasks with a precedence relationship among tasks. Each task has a processing time and needs resources to be completed. There are a number of different types of resources with certain availabilities.

The project can have renewable resources or non-renewable resources. The same amount of each resource type is available in each unit time period of the project in renewable resources case. If a task consumes a certain amount of a resource type in some time period of the project, the available amount of this resource type is decreased by the consumption amount of the task for the rest of the time periods of the project in non-renewable resources case. The project in this study is assumed to have renewable resources.

The project can be in single-mode or multi-mode. In multi-mode, tasks of the project have different processing times and need different amount of resources depending on the task mode. Tasks of the projects have more than one mode and mode of a task to be selected in the project is also determined to find the optimum schedule in the multi-mode case. The project type in this study is a single-mode resource constrained project scheduling problem so all tasks in the project have only one mode.

The aim of the RCPSP is to find a schedule that represents the start time of each task in the project so as to minimize the project completion time. The RCPSP is an NP hard problem (Blazewicz et al., 1983).

Single machine scheduling problem with sequence dependent setup times can be thought as the special case of the sequence dependent setup time flowshop problem (SDST flowshop) so we give the SDST flowshop problem definition. In SDST flowshop scheduling problem, there is a number of tasks that have to be processed sequentially on the machines. The objective is to find the ordering of the tasks on the machines that minimizes the completion time of all tasks. Processing order of the tasks can be different on each machine. In sequence dependent setup time flowshop problem, setup time of the machine for a task depends on the task previously processed on the machine. There is also a setup time for the task which is first processed on the machine. Sequence dependent setup time flowshop scheduling problem is among the difficult problems in scheduling theory. The single machine scheduling problem with sequence dependent setup times is NP hard as it is shown to be equivalent to the travelling-salesman problem (Pinedo M., 1995).

The integrated problem in this study, is a RCPSP, but there is additionally one special machine that some tasks need. Processing ordering of the tasks on this machine affects the setup time of the machine for a task that needs this machine. Tasks that need this special machine also need other resources. There is a precedence relationship among all tasks in the project as in the RCPSP problem. All tasks in the project need some resources to be completed as in the RCPSP and all resource types in the project are renewable.

In Chapter 2, we give a more detailed definition of the integrated problem, MIP model of the resource constrained project scheduling problem, MIP model of the single machine scheduling problem with sequence dependent setup times and MIP model of the integrated problem. In Chapter 3, literature review on the MIP modelling of the flowshop scheduling problem having sequence dependent setup times and metaheuristic methods to solve resource constrained project scheduling problem are discussed. In Chapter 4, a genetic algorithm for the integrated problem is developed and the parameter adjusting of the genetic algorithm is explained. In Chapter 5, the results of the genetic algorithm are compared with a hill-climbing-like search algorithm for the big problem instances. For the small problem instances, results of the genetic algorithms are compared with the MIP model results. We conclude in Chapter 6 that the genetic algorithm developed for the integrated problem is better than the hill-climbing-like search method and for the integrated problem, other metaheuristic solution methods can be developed.

CHAPTER 2

PROBLEM DEFINITION AND MODELLING

In this chapter, first, definition of the integrated problem is given. Next, the MIP model for the RCPSP and then single machine scheduling with sequence dependent setup times problem are provided. Last, MIP model for the integrated problem is developed.

2.1 Definition of the Integrated Problem

There is a set T of tasks such that $T = T_1 \cup T_2$, where T_1 is the set of tasks that need the machine and T_2 is the set of tasks that do not need the machine. Precedence relationships among tasks are the same as the precedence relationships in the RCPSP, and precedence relationships exists among all tasks in the project.

Setup time for a task j that needs the machine is denoted by S_{ij} that is setup time of the machine for task j which is immediately processed after task i on the machine. There is also a setup time for the task which is first processed on the machine. Setup of tasks are known and external, that is machine setup for a task is completed before the task becomes available to be proceessed on the machine, since the machine operator knows which task will be processed on the machine after a task is processed on the machine before the project starts and thus the machine operator can set up the machine immediately for the next task after a task is processed on the machine. Provided that the necessary resources are available, processing of a task that does not need the machine starts after all immediate predecessor tasks are completed. Provided that the necessary resources are available, processing of a task that needs the machine can start after the setup of the machine and all its immediate predecessors are completed.

Processing time of a task is denoted by P_i. Processing time of a task that needs the machine does not include the setup time. There are k types of resources and R_k denotes the available amount of resource k at every time period of the project timeline. Amount of resource k that task i needs is Q_{ik}.

Resource constraint is the same as the resource contraint in the RCPSP and all tasks need some resources. A task that needs the machine to be completed also needs some other resources. Similarly, a task that does not need the machine to be completed need some resources. A task that needs the machine to be completed use resources during the processing time only, but not during the setup time. The objective is the minimization of the completion time of the project.

If we think that there is no setup time for the machine in this problem, in other words, setup time is equal to zero, then the problem will be equivalent to the RCPSP. In this special case of the problem, the machine can be seen as a type of resource. We conclude that this special case of our problem is a RCPSP, therefore our integrated problem is NP hard. To explain some properties of the integrated problem, we use the example network diagram in Figure 1 below.

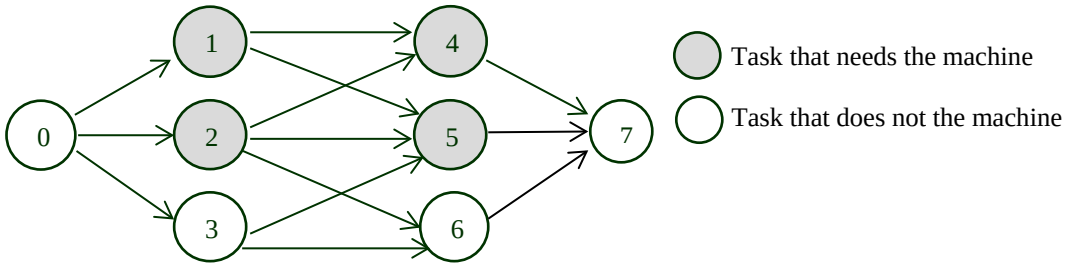


Figure 1 *A sample network diagram*

Task 0 and task 7 are dummy tasks. Completion time of task *i* is denoted by C_i. Start time of a task that needs the machine is after the setup of the machine is completed. Start time of task *i* is denoted by B_i.

Using the sample network diagram in Figure 1, T_1 = {1, 2, 4, 5} and T_2 = {3, 6}.

In any feasible solution, task 1 and task 2 have no predecessors and there is no precedence relation between task 1 and task 2 but task 1 or task 2 must be processed first on the machine and naturally the ordering of task 1 and task 2 occurs in any solution of the problem.

Suppose that in a feasible solution, processing order of tasks on the machine is 1-2-5-4. Task 1 is processed first on the machine.

In this feasible solution,

$C_1 \geq P_1 + S_{01}$
$B_2 \geq C_1 + S_{12}$
$B_4 \geq C_1$
$B_4 \geq C_2$
$B_4 \geq C_5 + S_{54}$
$B_5 \geq C_1$, $B_5 \geq C_2$, $B_5 \geq C_3$
$B_5 \geq C_2 + S_{25}$
$B_6 \geq C_2$
$B_6 \geq C_3$

Based on this solution before the project starts, the machine operator knows that after task 2 is processed on the machine, task 5 is processed on the machine, so the machine operator starts to set up the machine immediately after task 2 is processed on the machine. The machine operator does not wait for the completion of task 3 to set up the machine for task 5.

2.2 MIP Model for the RCPSP

Definition for the RCPSP is given in Chapter 1, so we give the MIP model according to this definition.
There are *n*-1 tasks in the project.

T is the set of tasks. *T* = {0, 1 . . . , *n*-1, *n* }. 0 and *n* are the dummy tasks. Task 0 represents the start task of the project and task *n* represents the end task of the project.

There are *s* types of resources. S = {0, 1 . . . , *s*-1, *s* }.

TP is the set of the unit time periods of the project. *TP* = {1 . . . , *m*-1, *m* }. Completion time of a heuristic method can be taken as a value for *m*. The result of the problem with the genetic algorithm is used to adjust the maximum completion time of the project in MIP modelling.

In Figure 2, an example of a project timeline is given. The project is completed at time 5 and starts at time 0. The numbers in the boxes are the numbers of time periods. Time period 3 means that (starting time point, ending time point) is (2, 3).

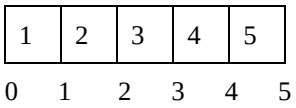


Figure 2 A project timeline

Parameters :

A_i : Set of immediate predecessors of task *i*

P_i : Processing time of task *i*. $P_0 = 0$ and $P_n = 0$

R_k : Available amount of resource *k* in each unit time period of the project

Q_{ik} : Amount of resource *k* that task *i* needs during a period of its processing time. $Q_{0k} = 0$ and $Q_{nk} = 0$

If task *j* is the immediate predecessor of task *i*, $IP_{ij} = 1$ else $IP_{ij} = 0$.

$$IP_{ij} = \begin{cases} 1 & \text{if } j \in A_i \\ 0 & \text{otherwise} \end{cases}$$

EC_i : Earliest completion time of task *i*

LC_i : Latest completion time of task *i*

Earliest completion time and latest completion times are calculated as in the critical path method.

Decision variables:

C_i : Completion time of task i,

$$Y_{it} = \begin{cases} 1 & \text{if task } i \text{ is completed at the end of time period } t, \\ 0 & \text{otherwise} \end{cases}$$

Y_{it} is defined in the interval $[EC_i - P_i, LC_i]$

The RCPSP model is explained below.

The objective is to minimize the completion time of dummy task *n.*

The RCPSP model:

$$\min\ C_n \qquad (1)$$

s.to.

$$\sum_{t=EC_i}^{LC_i} Y_{it} = 1 \qquad \forall\ i \in T \setminus \{0, n\} \qquad (2)$$

$$\sum_{t=EC_i-P_i}^{EC_i-1} Y_{it} = 0 \qquad \forall\ i \in T \setminus \{0, n\} \qquad (3)$$

$$\sum_{t=EC_i}^{LC_i} Y_{it}\, t = C_i \qquad \forall\ i \in T \setminus \{0, n\} \qquad (4)$$

$$C_i \geq C_j + P_i \qquad \forall\ i \in T,\ \forall\ j \in T,\ IP_{ij} = 1 \qquad (5)$$

$$\sum_{i=1}^{n-1} \sum_{\substack{t+P_i-1 \\ q=t \\ t\in[EC_i-P_i+1,\, LC_i]}} Q_{ik}\, Y_{iq} \le R_k \qquad \forall\, k \in S,\ \forall\, t \in TP \qquad (6)$$

$$Y_{it} \in \{0, 1\} \qquad \forall\, i \in T \setminus \{0, n\},\ \forall\ t \in [EC_i - P_i, LC_i] \qquad (7)$$

$$C_i \ge 0 \qquad \forall\, i \in T \qquad (8)$$

We explain the constraints below.
Constraint (2) guarantees that each task is completed at the end of a time period.

Constraint (3) guarantees that task *i* can not be completed at the end of a time period in the interval $[EC_i - P_i, EC_i - 1]$.

Constraint (4) specifies the completion time of each task using the assigned time period of the task in (2).

Constraint (5) ensures that if task *j* is the immediate predecessor of task *i*, completion time of task *i* is greater than or equal to the completion time of task *j*.

Constraint (6) ensures that at each time period and for each resource type, resource usage of all active tasks is less than or equal to the available resource amount. We say that a task is active at a time period if processing of the task is continuing at this time period. In constraint (6), to determine if a task is active in a time period *t*, the time periods $\{t, t+1, \ldots, t + P_i - 1\}$ are checked. If this set includes the time period, at the end of which the task is completed, the task is active at this time period.

Constraints (7) and (8) are the type and sign restrictions on the variables.

2.3 MIP Model for the Single Machine Scheduling Problem with Sequence Dependent Setup Times

This MIP model is the single machine case of the MIP model of the m-machine flowshop with sequence dependent setup times in the paper of Rios-Mercado and Bard (1997).

There are *n*-1 tasks in the problem.

T_0 is the set of tasks, $T_0 = \{0, 1 \ldots, n-1\}$. Task 0 is a dummy task which denotes the start and end task in the processing sequence of tasks.

T_1 is the set of tasks except the dummy task, $T_1 = \{1 \ldots, n-1\}$.

Parameters:

P_i : Processing time of task *i*. $P_0 = 0$

S_{ij} : Setup time of the machine when task *j* is immediately processed after task *i* for $i, j \in T_0$

There is also a setup time for the task which is first processed on the machine, in other words, $S_{0j} \neq 0$. The value of S_{j0} is equal to zero and does not affect the completion time of the project, but to construct the MIP model, we suppose that the value of S_{j0} is not equal to zero. The case of $S_{j0} = 0$ gives the same result with the case of $S_{j0} \neq 0$ because S_{j0} is not used in any time related constraint in the MIP model.

If *i* is equal to *j*, $S_{ij} = 0$

M is a big number.

Decision variables:

C_i : Completion time of task i

$$X_{ij} = \begin{cases} 1 & \text{if task } j \text{ is processed immediately after task } i \text{ on the machine,} \\ 0 & \text{otherwise} \end{cases}$$

X_{0i} = 1 means that task i is the first task processed on the machine

X_{i0} = 1 means that task i is the last task processed on the machine

C_{max} : Completion time of the project

The objective is to minimize the completion time of the project. Then the model is :

$$\min \ C_{max} \tag{9}$$

s.to

$$\sum_{\substack{j \in T_0 \\ S_{ij} \neq 0}} X_{ij} = 1 \qquad \forall i \in T_0 \tag{10}$$

$$\sum_{\substack{i \in T_0 \\ S_{ij} \neq 0}} X_{ij} = 1 \qquad \forall j \in T_0 \tag{11}$$

$$C_i + P_j + S_{ij} \leq C_j + M\ (1 - X_{ij}) \qquad \forall i \in T_1\ ,\ \forall j \in T_1 \tag{12}$$

$$C_i \leq C_{max} \qquad \forall i \in T_1 \tag{13}$$

$$P_i + S_{0i} \leq C_i + M\ (1 - X_{0i}) \qquad \forall i \in T_1 \tag{14}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i \in T_0\ ,\ \forall j \in T_0\ ,\ S_{ij} \neq 0 \tag{15}$$

$$C_i \geq 0 \qquad \forall i \in T_1 \tag{16}$$

Constraint (10) assigns an immediate successor to each task.

Constraint (11) assigns an immediate predecessor to each task.

Every task must have only one immediate successor and one immediate successor.

After assignment of immediate successor and predecessor by (10) and (11), there can be loops in the sequence.

We give examples for loops in Figure 3 and Figure 4 below.
Suppose that the project has 6 tasks. Task 0 is a dummy task.

Example 1 of loops is seen in Figure 3 below.

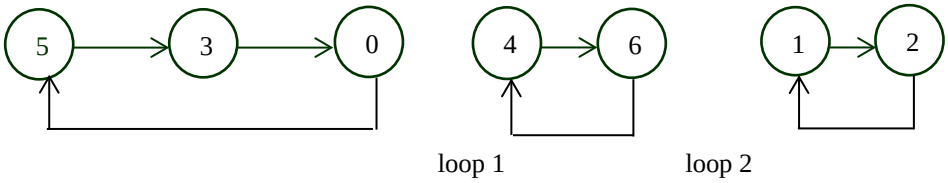


Figure 3 *First example of loops*

If task 0 (dummy task) is removed from the sequence, task 4 and task 6 are the successor and predecessor of each other in loop 1, and tasks 1 and 2 are the successor and predecessor of each other in loop 2.
This assignment of tasks is not a valid sequence.

In the second example for loops in Figure 4, if task 0 (dummy task) is removed from the sequence, all tasks except task 6 are the successor and predecessor of each other. This assignment of tasks is not a valid sequence.

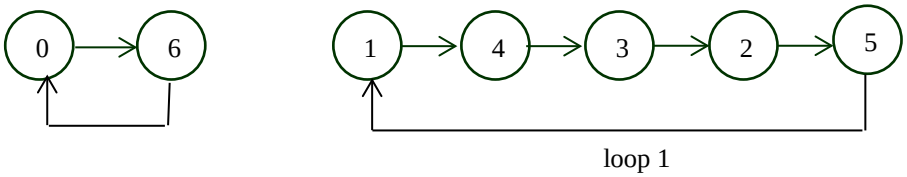


Figure 4 *Second example of loops*

A valid sequence of tasks is given in Figure 5 below.

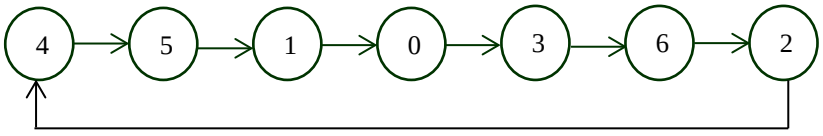


Figure 5 *An illustration of a valid sequence*

If task 0 (dummy task) is removed from the sequence, the sequence 3- 6- 2- 4- 5- 1 is left.
This assignment is a valid sequence. Task 3 is the first task which is processed on the machine.

Constraint (12) eliminates all the invalid sequences using the completion time of tasks except dummy task. Removing the dummy task is made possible in this constraint by using the set T_1 which does not contain the dummy task. If a task is in a loop, a completion time can not be assigned to this task by (12) because a contradiction occurs. When we look at example 1 of loops, we see that constraints $\{ C_6 \geq C_4 + P_6 + S_{46}, C_4 \geq C_6 + P_4 + S_{64} \}$ are produced by (12). There is no feasible completion time for task 4 and task 6, so this invalid sequence is eliminated by (12).

Constraint (13) takes the maximum of the completion times of tasks.

Constraint (14) ensures that if a task is the first task which is processed on the machine, lower bound for the completion time of this task is the processing time of this task plus initial setup time of machine for this task.

Constraints (15) and (16) define the decision variables.

2.4 MIP Model for the Integrated Problem

This model is the integration of the two models described in 2.2 and 2.3.

There are n-1 tasks in the project.

T is the set of tasks: $T = \{0, 1, \ldots, n\text{-}1, n\}$, where 0 and n are dummy tasks.
T_1 is the set of tasks that need the machine and $T_1 \subset T \setminus \{0, n\}$.
T_0 is the set of tasks that is the union of T_1 and $\{n+1\}$. $n+1$ is a dummy task which is used to determine the start and the end of the processing sequence of tasks on the machine.

S_{ij} : Setup time of the machine when task j is immediately processed
after task i for $i, j \in T_0$. There is also a setup time for the task which is processed first on the machine, in other words, $S_{(n+1)\,j} \neq 0$. We suppose that in the MIP model, $S_{j\,(n+1)} \neq 0$.
The meaning of X_{ij} is the same as X_{ij} in section 2.3.
There is a precedence relationship between all tasks and by looking the precedence relationship graph, we can reduce the number of decision variables X_{ij} where $j \in T_1$ and $i \in T_1$.
If i is equal to j, we don't define variable X_{ij} and $S_{ij} = 0$.
Now we give the definiton of a path.
A path in a graph is a sequence of nodes such that from each node there is an edge to the next node in the sequence. *Path*(i, j) is the set of all tasks in any path starting from i and ending at j.
If there is a task that is the member of T_1 and *Path*(i, j) \ $\{i, j\}$, we do not define the decision variable X_{ij} and $S_{ij} = 0$.
If task i is the successor of task j, we do not define the decision variable X_{ij} and $S_{ij} = 0$.

In the calculation of the earliest and latest completion time of task j that needs the machine, setup time for the task j is taken zero.

We do not explain other parameters and variables, because they are the same as those in section 2.2 and 2.3. Therefore MIP model of the integrated problem is given below.

The objective is to minimize the completion time of dummy task n, C_n.

The Integrated Model:

$$\min\ C_n \tag{17}$$

s.to.

$$\sum_{\substack{j \in T_0 \\ S_{ij} \neq 0}} X_{ij} = 1 \qquad \forall\, i \in T_0 \tag{18}$$

$$\sum_{\substack{i \in T_0 \\ S_{ij} \neq 0}} X_{ij} = 1 \qquad \forall\, j \in T_0 \tag{19}$$

$$C_i + P_j + S_{ij} \leq C_j + M\ (1 - X_{ij}) \qquad \forall\, i \in T_1\ , \forall\, j \in T_1\ , S_{ij} \neq 0 \tag{20}$$

$$P_i + S_{(n+1)\,i} \leq C_i + M\ (1 - X_{(n+1)\,i}) \qquad \forall\, i \in T_1 \tag{21}$$

$$\sum_{t=EC_i}^{LC_i} Y_{it} = 1 \qquad \forall\, i \in T \setminus \{0, n\} \tag{22}$$

$$\sum_{t=EC_i}^{LC_i} Y_{it}\, t = C_i \qquad \forall\, i \in T \setminus \{0, n\} \tag{23}$$

$$\sum_{t=EC_i - P_i}^{EC_i - 1} Y_{it} = 0 \qquad \forall\, i \in T \setminus \{0, n\} \tag{24}$$

$$C_i \geq C_j + P_i \qquad \forall\, i \in T\ ,\ \forall\, j \in T,\ \ IP_{ij} = 1 \tag{25}$$

$$\sum_{i=1}^{n-1} \sum_{t \in [EC_i - P_i + 1,\, LC_i]}^{t+P_i-1} Q_{ik}\ Y_{iq} \leq R_k \qquad \forall\, k \in S\ ,\ \forall\, t \in TP \tag{26}$$

$$Y_{it} \in \{0, 1\} \qquad \forall\, i \in T\ ,\ \forall\ t \in [EC_i - P_i, LC_i] \tag{27}$$

$$C_i \geq 0 \qquad \forall\, i \in T \tag{28}$$

$$X_{ij} \in \{0, 1\} \qquad \forall\, i \in T_0\ , \forall\, j \in T_0\ \ , S_{ij} \neq 0 \tag{29}$$

Constraints (18) and (19) construct a processing sequence on the machine for tasks that need machine.

The constraint (20) eliminates loops by using the completion time of tasks that need the machine. Loops can be formed by constraints (18) and (19)

Constraints (21), (22), (23), (24), (25), (26) have the same meaning as those in section 2.2 and in section 2.3.

Constraints (27), (28), (29) define the decision variables.

CHAPTER 3

LITERATURE REVIEW

In this chapter, first, we review the papers related to the flowshop scheduling problems with sequence dependent setup times for the MIP modelling of the integrated problem, because single machine scheduling problem is the special case of the flowshop scheduling and different MIP models in SDST flowshop scheduling literature can be found to model the single machine scheduling with sequence dependent setup times. Next, papers related to the project scheduling with sequence dependent setup times are reviewed. Last, papers related to metaheuristic solution methods for resource constrained project scheduling problems are reviewed.

3.1 Flowshop Scheduling Problems with Sequence Dependent Setup Times

Rios-Mercado and Bard (1997) provide a MIP model for m-machine flowshop scheduling problem with sequence dependent setup times. This model is the traditional TSP-based formulation. The model, which is given in section 2.3 of this thesis, is the single machine case of this model. The validity of the model does not depend on whether the sequence dependent setup times satisfy or do not satisfy the triangular inequality. Main decision variable in this model is based on the idea of immediate predecesor and immediate successor of a task in processing sequence of tasks.

Srikar and Ghosh (1986) propose a MIP model. Main decision variable in this model is based on the idea that a task is scheduled any time before another task. This model uses a less number of binary variables and constraints than the traditional TSP-based formulation. As long as the setup times satisfy the triangular inequality, this model is valid.

Stafford and Tseng (2001) provide two MIP models for the problem. One of the two models is based on the Srikar and Ghosh (1986) model and the other model uses a decision variable which determines the position of a task in the processig sequence of tasks. Each task can be assigned to only one position in the sequence.

There is also a different version of the flowshop problem. This flowshop problem is called hybrid or flexible flowshop problem with sequence dependent times. In hybrid flowshop problem (HFP), there is a sequence of stages and in each stage, there are parallel machines. Some stages can have one machine, but in HFP, there is at least one stage having more than one machine. Each task is processed by at most one machine in a stage. Some tasks can skip some stages, in other words, some tasks need not to be processed in all of the stages. Modelling of the hybrid flowshop problem has similar properties with the classical flowshop problem, so we give some literature for the modeling of HFP

Zandieh et al. (2005) propose two MIP models for the HFP: the first model is based on the TSP like model of Rios-Mercado and F. Bard (1997) and the second model is based on the position assignment of tasks in processing sequence on the machine like Stafford's model. First, a task is assigned to one of the parallel machines in stages then processing sequence of tasks on the machine is determined in these models.

Ruiz et al. (2008) propose a MIP model for a more realistic case of HFP. In this realistic case, there is also a precedence relationship among tasks. Anticipatory and non-anticipatory setup times are considered. In non-anticipatory setup time, the task should be available on the machine for the setup of the machine. In anticipatory setup time, if we know the next task to be processed on the machine, we can start the setup of the machine as the processing of the current task is completed. A TSP based formulation is proposed by them.

Ruiz and Maroto (2006) propose a genetic algorithm for the hybrid flowshops with sequence dependent setup times. Encoding of the solution representation in these problems is generally permutation of tasks in an array representation.

Many papers related with sequence dependent setup times are about single machine and parallel machine scheduling as well as flowshop and job shop problems. Sequence dependent setup times in project scheduling are mentioned in only a few papers.

Now we review the studies related to resource constrained project scheduling with sequence dependent setup times.

3.2 Project Scheduling with Sequence Dependent Setup Times

Mika et al. (2006) give some properties of setup time in project scheduling such as how setup time is related to precedence constraints in the project. Three cases of setup times are mentioned. In the first case, setups depend on the precedence constraints; in the second case, setups do not depend on the precedence constraints, and in the last case, setups are partially dependent on the precedence constraints. If setups are dependent on the precedence constraints, no activity's setup can be done before completion of all immediate predecessor of the activity. If setups do not depend on the precedence constraints, setups can be done in any time after the resource is released by the previous task executed. Setups can be done before completion of the predecessors of the activity. In this thesis, setup does not depend on the precedence constraints. Setup is done after the resource is released by the previous task which can not be in the predecessor set of the current task. In the last case, setups partially depend on the precedence constraints. For example, resource is necessary for a task that can depend on the output of the preceding task, in other words the output of the preceding task is transported to a place where the setup of the resource is done with this output.

Hartman and Briskorn (2010) give a survey of variants and extensions of resource constrained project scheduling problems. They provide a brief information about papers with setup time related resource constrained project scheduling problems.

Drexl et al. (2000) is the first paper that mentions on changeover time in RCPSP. In this paper, some tasks in the project have changeover times (sequence dependent setup times). Setup time related constraint is added to the MIP model of the RCPSP in a clever way. If there is a setup time S_{ij} between task i and task j, there must be at least S_{ij} plus (processing time of task j) time units between the completion time of task i and completion time of task j. In other words, at most one of tasks i and j must be scheduled in the time window $[t, t + S_{ij}]$. This idea is used in the modelling of the problem. For the validity of this MIP model, setup times must provide the property like triangular inequality. To produce the sequence 1-2-3 by this model, S_{12} plus processing time of task 2 plus S_{23} must be greater than or equal to $S_{13.}$ This problem is similar to the problem in this thesis, but modelling in this thesis uses TSP based approach and is developed with the help of single machine scheduling modelling. For the validity of the MIP model in this thesis, any relation between setup times is not necessary.

Schwindt and Trautmann (2000) provide a mathematical model for the RCPSP which consists of tasks having sequence dependent setup times. There are processing units (resources) and a task needs at most one of these processing units. All tasks need a processing unit. Setup of the processing unit depends on the sequence of activities which are scheduled on this processing unit. In this problem, output of a task's processing is the input for the immediate successor. A task needs some amount of the output of the predecessor. Naturally, precedence relation exists among tasks that have output-input dependency. They handle this precedence relation by the constraints on the minimum inventories of inputs instead of writing a direct precedence constraint in the model. They provide a mathematical model for this problem in which tasks are sequentially assigned to processing units. For example, there are two processing units and five tasks. On the one processing unit, tasks sequence is (3 - 5 - 4) and on the other processing unit, task sequence is (2-1). TSP based formulation is used to produce these sequences of tasks and for simplicity, setup time for the first task which is processed on a processing unit is assumed to be zero. They do not define any binary variable to handle other

resources (resource different from processing units) constraints, because they want to solve the problem by relaxing resource constraints.

There is no MIP model which is the same as the MIP model in this thesis to the best of our knowledge.

The main part of the problem is a RCPSP, so we investigate metaheuristic solution methods for the RCPSP to develop a genetic algorithm. Now, we review the literature for metaheuristic solution methods for the RCPSP.

3.3 Metaheuristic Methods for the RCPSP

In metaheuristic approaches, solution representation of the problem is very important. The solution representation is called as encoding of the solution. The solution representation must be transformed to a real schedule which has the time information of activities. This schedule generation process is called as decoding of the representation.

In the first phase of a metaheuristic method, initial solution(s) which is(are) encoded by a solution representation is(are) produced by a heuristic method or randomly. The main building blocks of a metaheuristic approach are solution representation types, solution decoding methods and initial solution(s) producing methods.

Sprecher et al.(1995) give the definitions for semi-active, active and non-delay schedules for the RCPSP.
Definition 1: A left shift of activity *j* is the one-period left shift if the difference between new start time of the activity *j* and old start time of the activity *j* is one.
Definition 2: A local left shift of activity *j* is a left shift of activity *j* which is obtainable by one or more successively applied one-period left shifts of activity *j*.
Definition 3: A local left shift of activity *j* is a left shift of activity *j* which is obtainable by one or more successively applied one-period left shifts of activity *j*.
In local shift of an activity, each intermediate derived schedules has to be resource and precedence feasible.
Definition 4: A global left shift of activity *j* is a left shift of activity *j* which is not obtainable by a local left shift.
Definition 5: A semi-active schedule is a feasible schedule where none of the activities can be locally left shifted.
Definition 6: An active schedule is a feasible schedule where none of the activities can be locally or globally left shifted.
The set of non-delay schedules is the subset of the set of active schedules, and the set of active schedules is the subset of semi-active schedules.

Hartmann and Kolisch (2000) give the definitions for the serial and parallel schedule generation scheme. They also give heuristic methods for RCPSP. Now, we mention these schedule generation schemes and heuristic methods.

Serial schedule generation scheme (SGS) consists of successive *n* steps; in each step an activity is selected from the eligible set and scheduled at the earliest precedence and resource feasible time. Eligible set is the set of unscheduled activities all predecessor activities of which have been scheduled in previous steps. The serial SGS produces active schedules. In scheduling problems with a regular performance measure such as makespan minimization, the optimal solution is the active schedule, therefore schedules produced by serial SGS contains the optimal solution.

In each step of the parallel SGS, activities in the eligible set are scheduled at the fixed time *t* until the eligible set is empty in the step. When the eligible set is empty, next step starts at *t_now* which is equal to the minimum completion time of scheduled activities that are completed after *t_previous* which is the time in the previous step. Eligible set is the set of unscheduled activities which have the property that if any one of them starts at time *t* of the step, the partial schedule preserves resource and precedence feasibility. Initial time which is the time in the first step is zero. The parallel SGS produces non-delay schedules. Optimal solution for a RCPSP with regular performance measure can

have the schedule which is not a non-delay schedule therefore, optimal solution can be the nonmember of the set of schedules produced by parallel SGS.

X-pass methods are construction type heuristics methods for the RCPSP problem and employ one or both of the SGS producing one or more schedules. In *X*-pass methods, selection of which activity in the eligible set depends on a priority rule or both priority rule and a selection probability. If a *X*-pass method produces one schedule, it is called as one-pass method. If it produces more than one schedule, it is called as multi-pass method.

In single pass methods, activities are selected according to a priority value which is assigned according to a priority rule in each iteration of a SGS. There are many researches on priority rules for RCPSP. Latest finish time is a well known priority rule.

Multi-pass methods are multi-priority methods, forward-backward scheduling methods and sampling methods. In multi-priority rule methods, different priority rules are applied on a SGS. Using convex combination of different priority rules, virtually unlimited number of priority rules can be generated. Priority value of activity *i* in the eligible set is obtained by the convex combination of priority values of different priority rules.

In sampling methods one priority rule and one SGS are used. Different schedules are produced by different selection probabilities assigned to activities in the eligible set. Sampling methods are categorized into three types (random, biased random, regret based biased random sampling).

In regret based biased sampling methods, different selection probabilities for an activity are produced by using different regret values. In computation of a selection probability, a regret value and priority value based computations are used. This is a biased method because by setting a proper value for the regret value in advance, an activity with the largest priority value in the eligible set can be selected.

In forward- backward methods, an SGS is used; and at each iteration, one of the forward SGS and backward SGS is applied alternatively. Forward SGS is the application of the SGS starting from start activity and backward SGS is the application of SGS starting from finish activity. Priority value for an activity is usully taken from start or the lastly produced schedule depending on which SGS(forward or backward) is applied. For example, start time of activities in the schedule which is produced by backward SGS in the previous iteration are taken as priority value for activities in forward SGS and activities in eligible set are selected in the same order with increasing order of priority values.

Klein (2000) gives priority rule based heuristics for resource constrained project problems. Serial forward /backward, parallel forward/backward schedule generation are explained. Bidirectional planning (construct schedule in forward and backward direction simultaneously) is given. Priority rules which are used for the selection of activity to be scheduled are given. Shortest processing time (SPT), Earliest Starting Time (EST), Greatest Resource Demand (GRD), Most Immediate Successors (MIS), Most Total Successors (MTS) are among the priority rules. Comparisons for different combinations of an SGS with a prioirity rule are given.

Kolisch and Hartmann (1999) give five solution representations (encodings) for the RCPSP.
These are activity list, random key, priority rule, shift vector and schedule scheme representations. An activity list is a sequence of activities where successor of activity *i* comes after activity *i*. An activity list can be decoded by a serial SGS which schedules the activities in the order of activity list. A random key representation is an array of random values which are assigned to activites. These random values are used to determine which activity is selected from the eligible set in an SGS. Priority rule representation is a one dimensional array of priority rules such as latest finish time, latest start time and shortest processing time. The size of this array is equal to the number of activities. In an SGS, *i*-th activity is selected for scheduling according to *i*-th priority rule in the array. When the *i*-th activity is scheduled, priority value of activities in the eligible set are calculated according to *i*-th priority rule and then one of the activities in the eligible set is selected according to these priority values. Shift vector representation is a one dimensional array of shift values which are used for time shifting of activities. In the decoding process, the *i*-th activity is started at time *t* which is the maximum completion time of the immediate predecessors of *i*-th activity plus shift value in the *i*-th position of the array. Resource constraint is not considered in the decoding process. This situation is

handled by adding penalty value to the objective function for the resource violations. Schedule scheme representation is a set of four relation sets which are conjuction, disjunction, parallelity and flexibility. If (i, j) is in the conjuction set, activity j can start when activity j is completed. A schedule that satisfies these relations may not be feasible. In the decoding phase, a heuristic method is used to construct a feasible schedule.

Mori and Tseng (1997) use a direct representation for their genetic algorithm, so decoding procedure is not necessary. In this direct representation, schedule order (a number) of activities and [start time, finish time] of activities are used. Schedule order determines the order activities are scheduled. [start time, finish time] of an activity is the start and the finish time of the activity in the feasible schedule.

Now we give metaheuristic methods that are used in solving the RCPSP to develop a genetic algorithm. Although the aim of this thesis is to develop a genetic algorithm, other metaheuristic approaches as well are investigated to obtain information for encoding, decoding and neighbour generation. First, genetic algorithms for the RCPSP are given, then other metaheuristic methods such as tabu search, scatter search and variable neighbourhood search are given.

3.3.1 Genetic Algorithms for the RCPSP

Hartmann (2002) uses activity list in his genetic algorithm and extra gene is added to the solution representation. This extra gene determines whether parallel or serial SGS is used for decoding. Initial population is constructed by a priority rule based sampling heuristic. Two-point crossover is used and by the mutation operator, an activity is right shifted to a perecedence feasible solution.

Hindi et al. (2002) use activity list and serial SGS in their genetic algorithm. Initial population is produced randomly. One-point, two-point and multi-point crossover operators are used. Randomly selected task is exchanged with a task. When the mutation operator is applied, the resulting activity list is precedence feasible.

Toklu (2002) uses direct representation in his genetic algorithm. Infeasible schedules can be generated. Penalty function is used for these infeasible schedules. Two different crossover operators are used. Randomly selected genes of father and mother are interchanged in crossover operator 1. Genes between randomly selected two positions are exchanged in crossover operator 2. By the mutation operator randomly selected gene is changed.

Alcaraz and Maroto (2002) develop a genetic algorithm for the multi-mode RCPSP. Activitiy list representation is used and additonally, forward(f) or backward(b) gene is used. Activitiy list with forward/backward gene is decoded by serial forward/backward SGS. In serial forward/backward (f/b) SGS, activities are scheduled starting fom the beginning/ending activity in the activity list. Initial population is generated by priority sampling method. Activity lists of the initial population are constructed according to which f/b gene is used. Activity list is filled from end/start to start/end when b/f gene is used. Two-point f/b crossover is used. First, two random points k_1 and k_2 ($k_1<k_2$) are selected. Then, first k_1 activities are taken from the mother's activity list when forward gene is used for mother and if backward gene is used for mother, activities starting from the positon k_3 which is greater than k_2 are taken from mother. Next activities between k_1+1 and k_2 are taken from father. Last, if forward gene is used for mother, activities starting from the positon k_3 which is greater than k_2 are taken from mother, and first k_1 activities are taken from the mother's activity list when backward gene is used for mother. In the mutation process, each activity is moved to a feasible position which is determined randomly.

Kochetov and Stolyar (2003) develop a genetic algorithm for the RCPSP. Solution representation is activity list. Path relinking is used for the crossover operator. Two solutions from the population are selected, and a path of solutions which connect these two solutions is constructed. One of the solution from the path is selected and tabu search is applied to it. The solution which is produced by the tabu search is added to the pool, then the worst solution in the pool is removed. After the new solutions of the pool are constructed, diversification is applied to the new pool. The neighbourhood structure which is used in tabu search is explained. To obtain a neighbour solution, the activity list is divided into three parts considering parallel activities. For the first part, serial SGS; for the second part, parallel SGS; and for the third part, serial SGS is used.

Gonçalves and Mendes (2003) use random key representation and parallel SGS with some modifications employed. In this modifed parallel SGS, a delay time for the activity's start time is considered. These delay times for activities are added to the random key representation.

Coelho and Tavares (2003) use activity list and serial SGS. A new crossover operator called late join function crossover is used for activity lists.
Tseng and Chen (2006) develop a hybrid metaheuristic method. Ant colony optimization, local search, genetic algorithm are used together. Precedence feasible activity list is used for decoding. The ants construct new activity list by looking for pheromones values. Pheromone value is the value of activity *i* which is assigned to position *j* in the list. If genetic algorithm search phase finds the best solution so far, pheromone values corresponding to this schedule are updated globally. In local search phase, first, forward schedule of activity list is generated. Second, backward schedule of activity list is generated. Third, forward schedule of the activity list of the backward schedule is generated. Last, best schedule of these schedules is selected. The best schedule is transformed to an activity list and kept. Global update and local update of the pheromone values are executed. Local updates of the pheromone values are applied when an ant constructs a new activity list. Gobal update is applied when all ants construct their activity list and the best activity is selected. When the genetic process stops, ant colony starts and then genetic process starts. Local search is applied to all activity list. In mutation operator, activities on the critical path of the schedule are determined and some of these activities are removed, then removed activities are inserted again to some positions in the activity list with precedence feasibility satisfied. Two mutation operators are applied.

Kim et al. (2006) develop an adaptive genetic algorithm for the RCPSP. In this genetic algorithm priority based encoding for activity priority is used. Activity having the high priority is first scheduled. Iterative hill-climbing method is used for local search. This procedure is implemented in each generation. Rate of the crossover and mutation operator are calculated at each generation. In calculation of the rates, average fitness values of parents and offspring at generation are used as the main factor. Adaptively regulated rates of crossover and mutation operator are used in the next generation production.

Valls et al. (2008) develop a hybrid genetic algorithm for the RCPSP. Activity list representation is used for encoding. Serial SGS is used for decoding. Initial population is generated by regret based biased sampling method with LFT priority rule. Double justification (first right justification, then left justfication) is applied to all members of the initial population. Right justification is the process of moving the activities to the right as much as possible in decreasing order of finish time of activities. Left justification is the process of moving the activities to the left as much as possible in increasing order of finish time of activities. Activity lists of the double justified schedules are produced. In parent selection first parent is the fittest individual and the second parent is randomly selected from the rest of the population. Peak cross-over operator used is based on the resource utilization in time periods. Activities that are scheduled in some part of the time period are taken into account. By the mutation operator, activitivies at succesive positions in the activitiy list are exchanged if precedence feasibility is retained. Next step starts after neigbour solutions from the best solutions in the pool are produced. Next generation is the pool of neighbour solutions. Neigbour solutions are constructed by the selection of the activity in the list order or the selection of the activity according to a biased random sampling in each iteration of SGS. Selection rule used depends on its probability.

Wuliang and Chengen (2009) develop a genetic algorithm in which priority-based encoding is used. Serial SGS is used for decoding. Position based crossover is used. By the mutation operator, two random positions are generated within the range [1,n], and if they are not equal to each other, then their priority values are exchanged.

Mendes et al. (2009) develop a genetic algorithm. Random key based representation is used in this genetic algorithm. Parameterized active schedule method is used to generate schedules. This schedule generation method produces parameterized active schedules that are the subset of active schedules. Non-delay schedules are the subset of the parameterized active schedules. By adjusting the parameter, the size of the set of parameterized active schedules can be reduced or increased. Chromosome representation consists of priority values and delay times. One parent for the crossover is selected from the fittest part of the population and the other parent is selected from the whole population. Randomly generated individuals are added to population instead of using a mutation operator.

Peteghem and Vanhoucke (2010) develop a bi-population genetic algorithm. Random key representation is used with topological ordering notation. Population 1 consists of left justified schedules and population 2 consists of right justified schedules. Backward procedure is applied on population 1 to build right justified schedules which are stored in population 2 and forward procedure is applied on population 2 to build left justified schedules which are stored in population1. Evoulation of the populations are done separately.
Montoya-Torres et al. (2010) develop a genetic algorithm in which activity list representation is used and serial SGS is used for decoding. Two crossover operators are used: one-point crossover and two point crossover operator. In mutation operator, subsequent positions are changed with each other with a certain probabilty.

When we look at the genetic algorithms, different encoding methods are used in genetic algorithms and some modifications of schedule generation schemes such as delay time addition to tasks' start time are applied. Local search methods such as iterative hill climbing and left (right) justification are applied to the population members. Hybrid genetic algorithms which are the use of genetic algorithms together with other metaheuristic algorithms such as ant colony optimization are developed. Adaptation of the crossover and mutation rates by using the information from the previous generation is done in some genetic algorithms.

3.3.2 Other Solution Methods for the RCPSP

Palpant et al. (2004) develop a local search method. First, forward-backward scheduling is applied to the schedule then some part of the schedule is improved considering parallel or sucessive activities. These steps are repeated.

Valls et al. (2003) develop a local search method. First, two types of moves are applied considering critical activities. Then random sampling is applied at time period which is determined from the schedule. Last, from the best solutions obtained from the previous step, neighbour solutions are produced. Neigborhood is constructed by the selection of the activity in the list order or the selection of the activity according to a biased random sampling in each iteration of SGS. Selection rule used depends on a probability.

Bouleimen and Lecocq (2003) develop a simulated annealing algorithm for the RCPSP and for multi-mode version. Activity list is used for encoding. A modificaiton of serial SGS is used for decoding. In this serial SGS, an activity *i* starts at fixed time *t* if precedence and resource feasibilty is attained. If any feasibility is not satisfied, the fixed time *t* is increased to a minimum value. When activity *i* starts at this new fixed value, resource and precedence feasibilty is satisfied in the partial schedule. If the activity is scheduled at fixed time t, the fixed time *t* is not changed. This serial SGS makes time increment when the activity can not start at time *t* of the step. Scheduling order of activities is the activity list order. Initial fixed time *t* is zero. Neighbour generation process is such that first an activity is selected randomly from the activity list. Then, the selected activity is moved to a position *i* which is within the [position *j* of the latest predecessor of selected activity, position *k* of the earliest successor of the selected activity]. Cyclical shifting of activities is applied.

Valls et al. (2005) use forward-backward improvement with the simulated annealing algorithm. Neigborhood is constructed by the selection of the activity in the list order or the selection of the activity according to a biased random sampling in each iteration of SGS. Selection rule used depends on a probability.

Thomas and Salhi (1998) use direct representation for the encoding of the solution in their tabu search algorithm. Infeasible schedules can be generated and these infeasible schedules are turned into a feasible schedule by a repair procedure.

Klein (2000) use activity list and serial SGS in a tabu search algorithm. For the neighborhood, two activities are exchanged if feasibility is satisfied.

Artigues et al. (2003) apply their insertion technique in a tabu search algorithm. Iteratively, an activity is selected and reinserted into a position with a network-flow based insertion algorithm in this insertion technique.

Nonobe and Ibaraki (2002) use activity list and serial SGS in their tabu search algorithm. Shifting activities and a neighborhood reduction are used.

Merkle et al.(2002) develop an ant colony optimization algorithm. An ant coressponds to how the serial SGS is applied when selecting an activity from the eligible set. An activity from the eligible set is selected by a decision which is given by using the combined information of latest start time priority rule and pheromone value. Pheromone value is the favorability of putting activity *i* immediately after activity *j* in the list. Forward or backward scheduling is also determined by additional ants. At the end of one step, local search is also applied.

Fleszar and Hindi (2004) develop a variable neigborhood search for the RCPSP. Solution representation is activity list representation and ten neighbourhood structures are used. Neighbourhood *k* is moving of randomly selected *k* activities to feasible points in the activity list. Maximum value of *k* is ten. They develop a moving strategy to move an activity to a new position. Initial solution is produced by priority rule sampling method and restricted neighbour search is applied to initial solution to get local optimum. The restricted neighbour search is also applied to intermediate solutions.

Muller (2009) develops an adaptive large neighborhood algorithm. Activity list is used as the encoding of the solution and serial SGS is used for decoding. Destroy(D) and repair(R) neighborhoods are defined. There are ten destroy and eleven repair neighborhoods. (D) removes some activities which are determined by a destroying rule. (R) inserts the removed activities by a repair rule. For example, (D) removes the activities in the high resource utilization time period. (R) inserts the activities according to a priority rule. Selection of which destroy or repair neighborhood is used in each iteration is done by using the past information about perfomance of the neighborhoods.

Chen (2011) develops a partical swarm optimization. Serial SGS on activity list representation is used and justification is used. Forward and backward scheduling is used. Each solution represents the position of a particle. Every particle goes to a new position in each step of the algorithm. The velocity of the particle determines the position. Velocity is determined by the best solution of all particles and local solution of each particle.

Debels et al. (2006) develop a scatter search for the RCPSP. Solution representation is topological order (TO) representation. TO representation is a sequence of numbers which are assigned to activities in the schedule. If start time of activity *i* in the schedule is less than start time of activity *j*, the value in the position *i* of the TO sequence is less than the value in the position *j* of the TO.

Paraskevopoulos et al. (2012) develop a scatter search based algorithm. A new solution representation scheme is used. This representation is called event list based representation. In this event based representaion, there are sets of tasks having the same start or finish time in the schedule. Adaptive iterated local search is used for improvement.

Kolisch and Hartmann (2006) give brief explanations of the papers about metaheuristic methods for the RCPSP.

There are some other papers about the new applications of some metaheuristic approaches such as immune algorithm, neurogenetic algorithm and some mixed applications of metaheuristic methods for the RCPSP (Paraskevopoulos et al., 2012).

We have included the other metaheuristic algorithms for the RCPSP, because the solution encoding and neighbourhood structures in these algorithms can be used in a genetic algorithm. In our study, we develop a genetic algorithm for the RCPSP having a single machine with sequence dependent setup times using an elitist parent selection method and activity list representation which is more suitable for handling sequence-dependent setup times. Sequence of tasks on the machine in our integrated problem corresponds to the sequence of tasks in the activiy list. We use the neighbourhood structure in the simulated annealing algorithm of Bouleimen and Lecocq (2003) as a mutation operator in our genetic algorithm. This mutation operator is suitable for changing the sequence of tasks on the machine to a sequence having less total setup time. Details of the genetic algorithm for the integrated problem is given in the next chapter.

CHAPTER 4

THE PROPOSED APPROACH: A GENETIC ALGORITHM

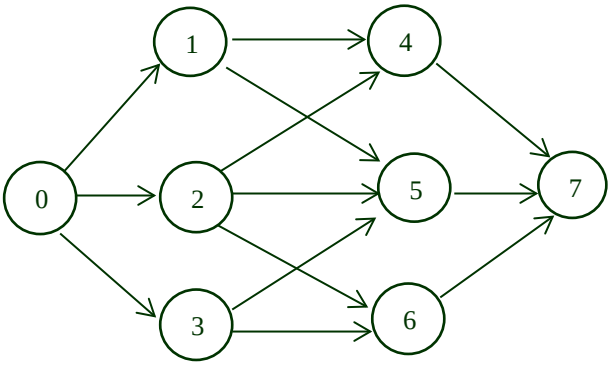
In this chapter, a genetic algorithm is proposed for the RCPSP including a single machine with sequence-dependent setup times. Some parameters of the algorithm are adjusted using small-sized problem instances.

4.1 The Genetic Algorithm

Genetic algorithms are the evolutionary algorithms that are inspired by the natural evolution. A genetic algorithm usually consists of initial population, selection, crossover, mutation and inheritance parts. Now, we give the detailed explanations for these parts of the genetic algorithm which is developed for the problem addressed.

4.1.1 Solution Representation (Encoding)

As a solution representation, one activity list which consists of all tasks in the problem is used. Activity list is a precedence-feasible list. If task *i* is a successor of task *j*, task *i* comes after task *j* in the activity list. Precedence graph of a problem can have many different activity list representations. An example network and one possible activity list representation are given below.



0	2	1	4	3	6	5	7

→ A possible activity list representation

Figure 6 *Example network and an activity list representation*

4.1.2 Decoding of the Solution Representation

Solution representation is decoded into a schedule by a modified serial schedule generation scheme (serial SGS) procedure.

The modified serial SGS is explained below.

Tasks to be scheduled by the serial SGS are selected in the same order as the order in the activity list. If task *i* is in the *k*-th position in the activity list, then task *i* will be the *k*-th scheduled task by the serial SGS.

Let
C_i: completion time of task *i* in the schedule.
J(i): task *i*'s immediate activity list predecessor that needs the machine.

I_i: set of immediate precedence graph predecessors of task *i*.
S_{ij}: setup time of the machine if task *j* is processed after task *i*.

If task *i* does not need the machine, it is started at time *t* where $\{t \geq C_j$ for all $j \in I_i$ and resource feasibility satisfied$\}$.

If task *i* needs the machine, it is started at time *t* where $\{t \geq C_j$ for all $j \in I_i$ and $t \geq C_{J(i)} + S_{J(i)\,i}$ and resource feasibility satisfied$\}$

4.1.3 Initial Population Generation

Initial population is generated by serial SGS which selects the activities from the eligible set randomly. In the serial SGS, we do not compute any time information for the task. We only put the tasks in the activity list.

4.1.4 Fitness function

Fitness of a solution is the completion time of the schedule. When the fitnesses of two solutions are equal, the solution having the higher total setup time can be better, because by decreasing the total setup time of the solution having the higher total setup time, there is a chance to decrease the completion time of the project.

4.1.5 Parent Selection

For the parent selection, the population is first sorted in increasing order of fitness value and then sorted in decreasing order of the total setup time of the solution. One parent is selected from the fittest part of the population randomly and the other parent is selected from the whole population randomly. The fittest part of the population, which is *x* percent of the population, has the fittest members. The value of *x* is adjusted to obtain better results. Mendes et al. (2009) use this parent selection procedure in the genetic algorithm.

4.1.6 Crossover Operation

For the crossover operation, one-point and two-point crossovers are used for the RCPSP in genetic algorithms based on activity list representation in some of the studies. We also test three-point crossover for this integrated problem. Based on the preliminary results, one-point crossover is selected, because it gives relatively better results for the test problems. One-point crossover is better to convey the good properties of the parents to the children for our integrated problem.

Test results for each crossover operator is given in Appendix B.

After selecting the one-point crossover, the genetic algorithm is further developed on the base of one-point crossover. These further developments include setup improvement and fitness improvement procedures. While determining the crossover operator type, the algorithm does not have the functionality of setup improvement and fitness improvement.

Setup improvement and fitness improvement procedures are explained after the one-point crossover explanation which is given below.

In one-point crossover, the random number R_1 is selected randomly in the range of [1, task count]. In child 1, first R_1-1 tasks are the mother's first R_1 tasks. These R_1-1 tasks are in the same order as the order in the mother's activity list. Tasks in the range [position R_1, task count] are taken from the father in such a way that a task which has the initial position in the activity list of the father among the tasks that are not in the child 1 is taken in each step. In child 2, mother is replaced with father and the same procedure is applied.

After applying one-point crossover, activity lists of children are precedence-feasible, because task order in the activity list of parents is preserved in the activity list of the child by the order preservation property of the one-point crossover.

An illustration of one-point crossover is given below in Figure 7.

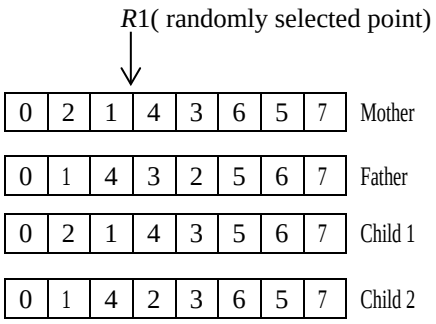


Figure 7 *An illustration of one-point crossover*

In child 1, tasks 0, 2 and 1 are directly taken from the mother and the remaining tasks are taken from the father in the same order as the activity list of the father.

In child 2, tasks 0, 1 and 4 are directly taken from the father and the remaining tasks are taken from the mother in the same order as the activity list of the mother.

After applying one-point crossover operation, setup improvement and fitness improvement procedures are also applied to the children. Now we explain setup improvement and fitness improvement procedures.

In setup improvement procedure, tasks that need the machine are considered to decrease the total setup time of child 1 when the total setup time of child 1 is greater than the mother's setup time. The idea behind the setup improvement is to take the good properties of the better solution if these properties make the receiver solution better. This idea can be used in other metaheuristic methods as a future study.

An illustration of the setup improvement procedure is given below in Figure 8.

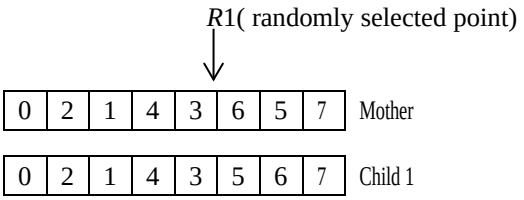


Figure 8 *An illustration of the setup improvement procedure*

Suppose that the total setup time of child 1 is greater than the mother's setup time. The setup time of child 1 is tried to be improved by making the order of the tasks similar to the order in the mother. Tasks which are tried to be made similar to the mother are the tasks that need the machine and are on the right side of the randomly selected point *R*1. Left side of the randomly selected point *R*1 in child 1 is the same as the mother.

Suppose that tasks 6, 5 and 7 are the tasks that need the machine on the right side of the randomly selected point *R*1. In child 1, task 6 is tried to be put on the left of task 5, or task 5 is tried to be put on the right of task 6 if precedence feasibility allows, and the total setup time of the child is improved. There will be no task that needs the machine and that is between task 6 and task 5 after this procedure is applied. Then task 5 is tried to be put on the left of task 7 or task 7 is tried to be put on the right of task 5 if precedence feasibility allows and the total setup time of the child is improved. There will be no task that needs the machine and that is between task 5 and task 7. This procedure is applied until the end of the activity list of the mother is reached. For child 2, the mother is replaced with the father and the same procedure is applied to child 2. When we apply this setup improvement procedure, we see that the results of the genetic algorithm are improved. After the setup improvement procedure, fitness improvement procedure is applied to child 1 if the fitness of child 1 is greater than the fitness of the mother after the application of the setup improvement procedure to child 1.

In fitness improvement procedure, a randomly selected task that is at the position in the range [*R*1, task count] is put in a precedence-feasible position if the fitness of child 1 is improved. If the fitness of the solution is not improved, randomly selected task is also put in a precedence-feasible position with a probability of 0.05. This procedure is applied until a number of tasks (randomly selected between 3 and 5) is put in a precedence-feasible position.

Children that are formed by one-point crossover operator, improved by setup and fitness improvement procedures and mutated, are conveyed to the next generation.

4.1.7 Mutation Operation

Mutation procedure consists of two steps. In the first step, a task that needs the machine is removed from the activity list and reinserted in the most setup improving and precedence-feasible position in the list. If there is no setup improvement when we put the task in any feasible position in the activity list, the task is also put in the least setup deteriorating and precedence-feasible position with a probability of 0.05.

In the second step, a task which needs the machine or does not need the machine is removed from the list and reinserted in a feasible position in the list, if there is a fitness improvement. If there is no fitness improvement when we put the task at the randomly selected feasible position in the activity list, the task is also put in this position with a probability of 0.05. In the second step, if we increase

the probability of putting the task at the randomly selected feasible position when there is no fitness improvement, the convergence of the population will be in later generations.

First step and second step are applied sequentially until a number of tasks (randomly selected between 3 and 5) are removed and tried to be inserted in each of the first and the second steps.

The insertion procedure can be defined as follows: task *i* is inserted randomly in a position which is in the range [R_1+1, R_2-1] and is different from the position of task *i*. R_1 is the position of the first list predecessor of task *i*. R_2 is the position of the first list successor of task *i*. Insertion is done by cyclical shifting of tasks.

An example of the application of the insertion procedure is given below in Figure 9.

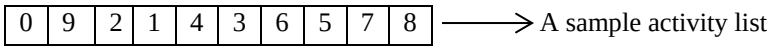


Figure 9 *An activity list representation*

Suppose that task 4 is removed and reinserted, task 7 and task 8 are the successors of task4, and task 0 and task 9 are the predecessors of task 4.

To find the first list predecessor of task 4, starting from task 4, the list is scanned from the right to the left until a predecessor is found. The first list predecessor of task 4 is task 9.

To find the first list successor of task 4, starting from task 4, list is scanned from the left to the right until a successor is found. The first list successor of task 4 is task 7.

Suppose that the position of task 5 is selected to insert task 4 (Figure 10).

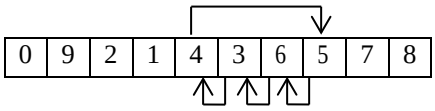


Figure 10 *First example of cyclical shift*

In the cyclical shift, task 3, task 6 and task 5 are shifted to the left. Task 4 is put in the previous position of task 5.

Suppose that the position of task 2 is selected to insert the task (Figure 11).

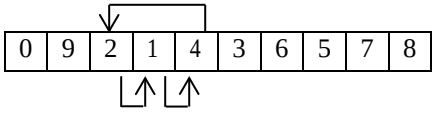


Figure 11 *Second example of cyclical shift*

In the cyclical shift, task 2 and task 1 are shifted to the right. Task 4 is put in the previous position of task 2.

Bouleimen and Lecocq (2003) use this move for the neighborhood in their simulated annealing algorithm.

At the end of the cyclical shift procedure, the precedence feasibility of the list is retained.

This insertion procedure is also applied when we apply setup improvement and fitness improvement to the children which are formed by one-point crossover operator.

4.1.8 Formation of the Next Generation

P_1 is the current population and P_2 is the next population which is used in the next iteration of the genetic algorithm.

To explain the next generation formation, P_1 and P_2 are divided into two sets.

P_{11} contains the fittest members of population P_1 and is equal to the fittest parents which are used in crossover operation:

$P_1 = P_{11} \cup P_{12}$
$P_2 = P_{21} \cup P_{22}$ and $P_{21} = P_{11,}$ P_{22} = Mutated children,

P_{22} is the set of mutated children which are produced by applying mutation on children which are formed after applying one-point crossover, setup and fitness improvement procedures.

4.2 Parameters of the Genetic Algorithm

Parameters of the genetic algorithm are adjusted to a better value by testing some values for the parameters.

4.2.1 Population Size (parameter 1)

Population size depends on the number of tasks in the problem.
Population size is $C \times$ (number of tasks in the problem). C is a constant number. Tested values for C are 1, 3 and 5. $C = 5$ is taken in the genetic algorithm according to test results.

4.2.2 Percentage Value for the Fittest Part (parameter 2)

One parent is always selected from the fittest part of the population. This fittest part is *X* percent of the population. Tested values for *X* are 10, 20 and 30.

4.2.3 Generation Number (parameter 3)

Generation number determines when the genetic algorithm stops. Values tested for the generation number are 50, 100, 150, 200, 250. Maximum number of tasks in the problems tested is 120.

4.2.4 Number of Tasks Selected for Mutation (parameter 4)

Two children are produced by the crossover operation. These children are improved, mutated and then are conveyed to the P_{22} of next population. The number of tasks selected for the mutation in a child is randomly determined in the range [3, 5] in each of the first and second step of the mutation procedure.

Fleszar and Hindi (2004) develop a variable neighborhood search algorithm and maximum number of tasks that are removed and inserted is adjusted to 10. The intervals for the number of tasks selected for mutation are determined from this experience.

4.3 Adjusting the Parameters Using the Test Values

To adjust the parameters, two problem sets are created. Problems are taken from the PSPLIB website (http://webserver.wi.tum.de/psplib/).
One problem set contains two problems having 30 tasks and the other problem set contains two problems having 60 tasks. Also, some tasks that need the machine are selected in a problem. These tasks use the machine for processing. A problem has the property that 70 % of tasks use the machine. Setup time S_{ij} is taken randomly in the range [1, processing time of task j-1].

Problems are grouped according to their network complexity and resource complexity.

Network complexity (NC) is $\dfrac{\text{number of arcs in the network}}{\text{number of nodes in the network}}$.

Resource complexity (RC) is $\dfrac{\sum_i \dfrac{\text{amount of available resource } i}{\text{total resource } i \text{ need of tasks}}}{\text{number of resource types}}$.

The higher value of network complexity means that there are more precedence relationships among tasks in the project and sequencing alternatives of the tasks that need the machine are less because of obeying of the sequence of tasks to the precedence relationships. If the value of the resource complexity of a project is smaller, there will be a higher competition among tasks for the resources.

Problem code and problem properties are given in the Table 1 below.

Table 1 *Problems used in parameters adjusting*

Problem	Number of tasks	NC	RC	*X*% of tasks that use the machine
j3010_4	30	1.5	0.17	70
j3045_4	30	2.12	0.10	70
j6010_4	60	1.5	0.11	70
j6045_4	60	2.11	0.05	70

The problem j3010_4 has less value for the resource complexity than the problem j3045_4, so in the scheduling of the tasks of the problem j3010_4, there will be more resource usage conflicts among tasks. When the number of tasks that need the machine increases, alternatives for the sequences of tasks increase and selection of the best sequencing of tasks becomes more difficult.

4.3.1 Adjusting of Parameter 1 (Coefficient *C* in Population Size Calculation)

When we fix the value of parameter 2 (percentage value for the fittest part in parent selection) and parameter 3 (generation number) at some tested values which are mentioned in sections 4.2.2 and 4.2.3, the value of 5 for parameter 1 gives better results than other values tested for parameter 1.

Ten replications for each problem and parameters combination are done. Average results of these ten replications for each problem with some parameters combination are given in average result column of the tables below (Tables 2 thru 9).

Table 2 *Average results for problem j3010_4*

Parameter 1	Parameter 2	Parameter 3	Average result
1	10	100	177
3	10	100	175.7
5	10	100	175.1
1	20	150	176.3
3	20	150	175.2
5	20	150	175
1	30	200	176.1
3	30	200	175.4
5	30	200	175.2

When we examine the first three rows of the table above, we see that the value for (parameter 2, parameter 3) is fixed at the value (10, 100). The value 5 for the parameter 1 gives average result 175.1, which is the average of the ten replications and is better than the average results of the other tested values, 1and 3, for the parameter 1.

The remaining rows of the table can be interpreted similarly. The similar interpretation can be done for the other problems (Tables 3, 4 and 5).

Table 3 *Average results for problem j3045_4*

Parameter 1	Parameter2	Parameter 3	Average result
1	10	100	148.7
3	10	100	148.2
5	10	100	148
1	20	150	148.5
3	20	150	148.1
5	20	150	148
1	30	200	148.4
3	30	200	148.1
5	30	200	148.1

Table 4 *Average results for problem j6010_4*

Parameter1	Parameter 2	Parameter 3	Average result
1	10	100	290.4
3	10	100	286
5	10	100	284.3
1	20	150	289.2
3	20	150	284.9
5	20	150	283.8
1	30	200	287.1
3	30	200	285
5	30	200	283

Table 5 *Average results for problem j6045_4*

Parameter 1	Parameter 2	Parameter 3	Average result
1	10	100	241.2
3	10	100	239.6
5	10	100	239.3
1	20	150	240
3	20	150	239.2
5	20	150	239
1	30	200	240.3
3	30	200	239.4
5	30	200	238.9

When we look at all tables, we see that value of 5 for parameter 1 is better than the values, 1 and 3, when parameter 2 and parameter 3 are fixed at some value.

4.3.2 Adjusting of Parameter 2 (Percentage Value for the Fittest Part in Parent Selection)

After selecting value '5' for parameter 1, we determine which tested value for parameter 2 gives a better result.

When we fix values of parameter 3 (generation number) at some tested values which are mentioned in section 4.2.3, value of '20' for parameter 2 gives better results than other tested values for parameter 2. Ten replications are done for each test problem.

In the first three rows of the Table 6 below, parameter 3 is fixed at the value 150. The value '5' for the parameter 1 is selected in the all rows of the table because the value '5' for the parameter 1 gives better results in adjusting of the parameter 1 in the previous part. In the last three rows of the table, parameter 3 is fixed at the value '200'.

When we look at the first and last three rows of the table, the value '20' for parameter 2 gives better results than the values '10' and '30' for parameter 2. In all tables (Tables 7, 8 and 9), the value 20 for the parameter 2 gives better results when the parameter 3 is fixed at some value which is given in the tables. Therefore, the value 20 for the parameter 2 is selected in the implementation of the genetic algorithm.

Table 6 *Average results for problem j3010_4*

Parameter 1	Parameter 2	Parameter 3	Average result
5	10	150	175
5	20	150	175
5	30	150	177
5	10	200	175.2
5	20	200	175
5	30	200	175.2

Table 7 *Average results for problem j3045_4*

Parameter 1	Parameter 2	Parameter 3	Average result
5	10	150	148.1
5	20	150	148
5	30	150	148.1
5	10	200	148.1
5	20	200	148
5	30	200	148.1

Table 8 *Average results for problem j6010_4*

Parameter 1	Parameter 2	Parameter 3	Average result
5	10	150	285.1
5	20	150	283.8
5	30	150	284.1
5	10	250	284.1
5	20	250	283.2
5	30	250	283.9

Table 9 *Average results for problem j6045_4*

Parameter 1	Parameter 2	Parameter 3	Average result
5	10	150	239.7
5	20	150	239
5	30	150	239.2
5	10	250	239.6
5	20	250	238.7
5	30	250	239

4.3.3 Adjusting of Parameter 3 (Generation Number)

After selecting value ‘5’ for parameter 1 and value ‘20’ for parameter 2, we determine which tested value for parameter 3 gives a better result. Ten replications are done for each problem set. Average results of these ten replications are given for each parameter setting (Tables 10, 11, 12 and 13). Best of the population is the fittest completion time among the completion times of population schedules. Average of the population is the average of the completion time of the schedules in the population.

Table 10 *Average results for problem j3010_4*

Parameter 1	Parameter 2	Parameter 3	Average result of the best of the population	Average result of the average of the population
5	20	1	199	217.4
5	20	10	178	187
5	20	20	176	186
5	20	30	175.5	184
5	20	40	175.3	181
5	20	50	175.2	178
5	20	100	175.1	177.6
5	20	150	175	176.8
5	20	200	175	176
5	20	250	175	176

Table 11 *Average results for problem j3045_4*

Parameter 1	Parameter 2	Parameter 3	Average result of the best of the population	Average result of the average of the population
5	20	1	164	178
5	20	10	149	155
5	20	20	148.6	152
5	20	30	148.5	151
5	20	40	148.4	150
5	20	50	148.2	149.5
5	20	100	148.1	149
5	20	150	148	149
5	20	200	148	149
5	20	250	148	149

For the problems j3010_4 and j3045_4, average result of the best of the population and average result of the population do not change after 150 generations. We can take 150 as a generation number for the small problem instances.

For the problems j6010_4 and j6045_4, average result of the best of the population and average result of the population decrease when the generation number increases up to the value of 250. The generation number 250 can be taken when the problem size grows. The value ‘250’ for the generation number is taken in the implementation of the genetic algorithm, because the value ‘250’ gives better results for all problem instances sized from smallest to largest. Maximum number of tasks in the problems is 120.

Table 12 *Average results for problem j6010_4*

Parameter 1	Parameter 2	Parameter 3	Average result of the best of the population	Average result of the average of the population
5	20	1	329	353
5	20	10	291	304
5	20	20	288	294
5	20	30	285	291
5	20	40	284.5	291
5	20	50	284.3	290
5	20	100	284.1	286
5	20	150	283.8	285.6
5	20	200	283.6	285.4
5	20	250	283.2	285.2

Table 13 *Average results for problem j6045_4*

Parameter 1	Parameter 2	Parameter 3	Average result of the best of the population	Average result of the average of the population
5	20	1	275	295
5	20	10	242	250
5	20	20	241	242
5	20	30	240	242
5	20	40	239.8	241
5	20	50	239.7	240.8
5	20	100	239.3	240.6
5	20	150	239	240.2
5	20	200	238.9	240.1
5	20	250	238.7	240

4.4 Robustness of the Genetic Algorithm

Robustness measures the independency between the inital population and quality of the result of the genetic algorithm. If there is a high independency between them, the algorithm is said to be robust.

To test the robustness of the algorithm, all problems are solved for 10 times with the selected parameter values and the minimum, maximum, average and standart deviation of the results of ten runs are given for each problem. Through examining these values, robustness of the algorithm can be evaluated.

The comparison for the performance of the algorithm can be possible with one run result of the algorithm if the algorithm is robust enough.

All problems and their properties are given in Table 14 below.

Table 14 *All problems and their properties*

Problem	Number of tasks	NC	RC	*X* % of tasks that use the machine
j1510_4	15	1.29	0.32	70
j1510_8	15	1.41	0.32	30
j1545_4	15	1.58	0.18	70
j1545_8	15	1.76	0.18	30
j3010_4	30	1.5	0.17	70
j3010_8	30	1.5	0.17	30
j3045_4	30	2.12	0.10	70
j3045_8	30	2.12	0.08	30
j6010_4	60	1.5	0.11	70
j6010_8	60	1.5	0.11	30
j6045_4	60	2.11	0.05	70
j6045_8	60	2.11	0.05	30
j9010_4	90	1.5	0.13	70
j9010_8	90	1.5	0.09	30
j9045_4	90	2.10	0.04	70
j9045_8	90	2.10	0.04	30
j12010_4	120	1.5	0.09	70
j12010_8	120	1.5	0.08	30
j12045_4	120	2.1	0.1	70
j12045_8	120	2.1	0.12	30

Minimum, maximum, average and standart deviation of the results of 10 runs are given in Table 15. The reader can refer to Appendix C for all results.

Table 15 *All problems and their 10 runs' results*

Problem	Minimum	Maximum	Average	Standart deviation
j1510_4	103	103	103	0
j1510_8	33	33	33	0
j1545_4	70	70	70	0
j1545_8	59	59	59	0
j3010_4	175	175	175	0
j3010_8	89	89	89	0
j3045_4	148	148	148	0
j3045_8	95	99	97	1.76
j6010_4	282	286	284.2	1.39
j6010_8	137	137	137	0
j6045_4	237	240	239	1.05
j6045_8	161	164	162.6	0.84
j9010_4	480	485	481.9	1.79
j9010_8	206	208	207	1.05
j9045_4	391	393	392.1	0.87
j9045_8	240	244	241.6	1.71
j12010_4	579	585	581.5	2.06
j12010_8	289	292	290.8	1.03
j12045_4	479	481	480	0.47
j12045_8	285	291	286.5	1.95

The algorithm gives results which are near the average value and the standart deviation is small with respect to the average value in all problems. When the initial solutions, which are produced randomly, changes from one replication to another replication, the results of the genetic algorithm are near to each other. Therefore, it can be said that the algorithm is robust.

4.5 Changes in Total Setup Time

To look at how the total setup time of the tasks that need the machine changes from one result of the genetic algorithm to the other result of the genetic algorithm, the problem j3045_8 is selected. Total setup time of a solution is the sum of the sequence dependent setup times in the solution sequence of the tasks that need the machine. For the problem j3045_8, we can compare the total setup time of the solution having fitness value of 95 with the total setup time of the solution having fitness value of 99.

Ten replication results of the genetic algorithm indicate that total setup time of the solutions having fitness value of 95 is 21 and total setup time of the solution having fitness value of 99 is 21or 22.

Total setup time of the solutions having fitness value of 99 are different according to the ten replication results of the the problem j3045_8.

Small difference between total setup time of the solution may or may not affect the fitness of the solution. In some cases, it may affect the fitness of the solution, because changing the order of a task in the sequence can affect the schedule of the other tasks because of resource constraints. Different solutions can have the same total setup times but different fitness values, because different sequence

of tasks can produce the same total setup time but these different sequences of tasks can affect the fitness of the solutions in a different way because of resource constraints.

CHAPTER 5

COMPARISON OF RESULTS

The results of the genetic algorithm is compared with MIP model results for small-sized problems and compared with a hill-climbing-like search algorithm for large-sized problems. The pseudo code for the genetic algorithm is given in Appendix A.

In each step of the hill-climbing-like search algorithm, one neighbour of the current solution is replaced with the current solution if the neighbour solution is better than the current solution. If the neighbour solution is worse than the current solution, the neighbour solution is replaced with the current solution with a probability of 0.5.

The neighbour in the hill-climbing-like search algorithm is the removal of a randomly selected task from the activity list and inserting it in a precedence-feasible position in the activity list. In calculation of the result of the hill-climbing-like search algorithm, 10 replications are done for each problem and the average of 10 replications is taken. CPU time for the each replication in the hill-climbing-like search algorithm is the CPU time of the genetic algorithm for each problem.

All program codes are written in Microsoft Visual C# .NET. Problems are the same as the problems in section 4.4.

5.1 Comparison with MIP Model

The number of problems which are used to compare the results are five, because MIP model needs very long runtime for the large-sized problems. One run is executed for the genetic algorithm, because the genetic algorithm is robust. The generation number of the genetic algorithm is taken as 100 for the problems having 15 tasks and as 250 for the problems having 30 tasks.

The problems and results for the genetic algorithm and MIP model are given in Table 16 below.

Table 16 *Problems solved via MIP model and Genetic Algorithm and comparison*

Problem	CPU Time of Genetic Algorithm (sec.)	Genetic Algorithm Result	CPU Time of MIP Model (sec.)	MIP Model Result
j1510_4	8	103	220	103
j1510_8	7	33	1.5	33
j1545_4	8	70	1335	70
j1545_8	7	59	24	59
j3010_8	87	89	2313	89

All results are the same as the MIP Model. This shows that the genetic algorithm gives correct results and performs well in solving small-sized problems. 70% of tasks of the problem j1510_4 need the

machine to be completed and 30% of tasks of the problem j1510_8 need the machine to be completed. The MIP model solves the problem j1510_4 in more time than the problem j1510_8. When the number of tasks that need the machine increases, the problem is difficult to be solved by the MIP model and the MIP model solves in long CPU time.

5.2 Comparison with a Hill-climbing-like Search Algorithm

The problems, and the results of both the genetic algorithm and the hill-climbing-like search algorithm are given in Table 17 below. In the genetic algorithm result column and a hill-climbing-like search algorithm result column of the table, average of results of 10 replications is givenCPU time for each replication of the hill-climbing-like search algorithm is the CPU time for the genetic algorithm and is given in the CPU Time (sec.) column of the table. The reader can refer to Appendix D for all results of the hill-climbing-like search algorithm.

Table 17 *Problems for the hill-climbing-like search algorithm comparison and results of them*

Problem	CPU Time (sec.)	Genetic Algorithm Result	A Hill-climbing-like Search Algorithm Result
j1510_4	25	103	110.1
j1510_8	19.5	33	34.7
j1545_4	22	70	72
j1545_8	20	59	62.1
j3010_4	84	175	181.1
j3010_8	87	89	94.2
j3045_4	80	148	153.9
j3045_8	77	97	104.2
j6010_4	337	284.2	298.8
j6010_8	308	137	144.7
j6045_4	304	239	248.2
j6045_8	393	162.6	174
j9010_4	871	481.9	501.8
j9010_8	763	207	221.3
j9045_4	785	392.1	412
j9045_8	850	241.6	257.1
j12010_4	1708	581.5	611.9
j12010_8	1529	290.8	310.7
j12045_4	1422	480	500.8
j12045_8	1476	286.5	308.5

When we look at the table above, genetic algorithm is better than the hill-climbing-like search algorithm for all problems. The maximum difference between the results of these algorithms is for the problem j12010_4. This difference, which is 30.4, is a big value when compared with the genetic

algorithm result. The difference between the result of the genetic algorithm and the hill-climbing-like search algorithm generally increases when the number of tasks in the problem increases.

5.3 Results for the Problem Set without the Machine

The problems, and the results for the genetic algorithm and best known values for the problem sets when no machine (crane) is needed for the task completion are given in Table 18 below. There is no task that needs the machine in the problem sets, so the genetic algorithm does not use any functionality related to setups. The problems to be solved are solely resource constrained project scheduling problems. Best known values for the problem sets are taken from the PSPLIB web site.

The best result of the 10 replications of the genetic algorithm is given under the column of best result of the 10 replications of the genetic algorithm column.

Table 18 *Problems and results for the no-machine case*

Problem	Best Result of the 10 Replications of the Genetic Algorithm	Best Known Value From the PSPLIB Web Site
j3010_4	58	58
j3010_8	54	54
j3045_4	84	84
j3045_8	94	94
j6010_4	80	80
j6010_8	65	65
j6045_4	109	108
j6045_8	132	129
j9010_4	94	94
j9010_8	81	81
j9045_4	140	135
j9045_8	164	160
j12010_4	95	95
j12010_8	114	114
j12045_4	103	103
j12045_8	103	103

When the problems have no tasks that need the machine, the result of the genetic algorithm is usually equal to the best known results for the problem sets. For the problem set j6045_4, j6045_8, j9045_4 and j9045_8, genetic algorithm result is not equal to the best known values. According to the PSPLIB web site, there is no known optimum solution for these problem set (j6045_4, j6045_8, j9045_4 and j9045_8). For the other problems in the table, best known values are equal to the optimum solution values.

CHAPTER 6

CONCLUSION

Sequence dependent setup times are usually considered in single machine scheduling, parallel machine scheduling, job shop scheduling and flowshop scheduling problems in literature. There are few studies about project scheduling with sequence dependent setup times. In a project, some resources can have setup times which depends on the sequence of activities that use this resource.

In this study, the single resource (machine) with sequence dependent setup times in resource constrained project scheduling problem is considered, mathematically modelled and solved with a genetic algorithm which is developed by using different ideas from literature. The MIP model of the problem is developed by the help of the single machine case of the MIP model of sequence dependent setup time flowshop scheduling problem and the help of MIP model of the resource constrained project scheduling problem. As far as we know, there is no genetic algorithm for the problem studied in this thesis. In the developed genetic algorithm, total setup time and fitness of the solutions are tried to be improved at the same time. The idea, which is used in the setup improvement phase of the genetic algorithm, is to take some good properties of the better solution if these properties make the receiver solution better. This idea can be used in other metaheuristic approaches to obtain better solution. The results of the genetic algorithm are compared against the solutions obtained by a hill-climbing-like search algorithm for the large-sized problems and by a developed MIP model for the small-sized problems. The results show that the genetic algorithm is always better than the hill-climbing-like search algorithm for all problems.

Future work may be on developing a metaheuristics method for project scheduling having parallel machines or flowshop structures with sequence dependent setup times. Moreover, different MIP modelling of these problems can be investigated or other metaheuristics like ant colony optimization, tabu search, simulated annealing methods can be developed for this problem and results can be compared with this study. Our genetic algorithm can be improved by producing initial population by a heuristic method rather than a random method. Different solution representations like random key representation can be used in a genetic algorithm for this problem.

REFERENCES

Alcaraz J. and Maroto C., 2002. A new genetic Algorithm for the multi-mode resource- constrained project scheduling problem. *Journal of the Operational Research Society 54, 614–626*

Artigues C., Michelon P. and Reusser S., 2003. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research 149: 249–267.*

Blazewicz J., Lenstra J.K. and Rinnooy Kan A.H.G, 1983. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics, 5 (1): 11-24.*

Bouleimen K. and Lecocq H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research 149: 268–281*

Chen R.-M., 2011. Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem. *Expert Systems with Applications 38: 7102–7111*

Coelho J. and Tavares L., 2003. Comparative analysis of metaheuricstics for the resource constrained project scheduling problem,.*Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal.*

Debels D., Reyck B.D., Leus R. and Vanhoucke M., 2006.A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research 169: 638–653*

Drexl A., Nissen R., Patterson J. H. and Salewski F., 2000. ProGen/px - An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *European Journal of Operational Research, 125: 59-72*

Fleszar K. and Hindi K.S., 2004. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research 155: 402–413*

Gonçalves J. and Mendes J., 2003. A random key based genetic algorithm for the resource-constrained project scheduling problem. *Technical report, Departamento de Engenharia, Universidade do Porto.*

Hartmann S. and Kolisch R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research 127: 394-407*

Hartmann S. and Briskorn D., 2010. "A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem". *European Journal of Operational Research, 207 (1): 1-14*

Hartmann S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics 49: 433–448.*

Hindi K.S., Yang H. and Fleszar K., 2002. An evolutionary algorithm for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation 6: 512–518.*

Kim K., Gen M. and Kim M., 2006. Adaptive genetic algorithms for multi-resource constrained project scheduling problem with multiple modes. *ICIC International 1249-4196: 41-49*

Klein R., 2000. Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects. *European Journal of Operational Research 127: 619-638*

Klein R., 2000. Project scheduling with time-varying resourceconstraints. *International Journal of Production Research 38 (16) : 3937–3952.*

Kochetov Y.A. and Stolyar A.A., 2003. Evolutionary Local Search with Variable Neighborhood for the Resource Constrained Project Scheduling Problem. *Workshop on Computer Science and Information Technologies CSIT'2003, Ufa, Russia, 2003*

Kolisch R. and Hartmann S., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research 174: 23–37*

Kolisch R. and Hartmann S., 1999. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. *Project scheduling: recent models, algorithms, and application: 147-178*

Mendes J.J.M., Gonçalves J.F. and Resende M.G.C., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research 36: 92 – 109*

Merkle D., Middendorf M. and Schmeck H., 2002. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation 6: 333–346.*

Mika M., Waligora G. and Weglarz. J., 2006. Modelling setup times in project scheduling. *Perspectives in Modern Project Scheduling, Springer: 131-163*

Montoya-Torres J.R., Gutierrez-Franco E. and Pirachican-Mayorga C., 2010. Project scheduling with limited resources using a genetic algorithm. *International Journal of Project Management 28: 619–628*

Mori M. and Tseng C.C., 1997. A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research 100: 134-141.*

Muller L.F., 2009. An Adaptive Large Neighborhood Search Algorithm for the Resource-constrained Project Scheduling Problem. *MIC: The VIII Metaheuristics International Confrence*

Nonobe K. and Ibaraki T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: C.C. Ribeiro, P. Hansen (Eds.*). Essays and Surveys in Metaheuristics, Kluwer Academic Publishers: 557–588.*

Palpant M., Artigues C. and Michelon P., 2004. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research, 131: 237–257.*

Paraskevopoulos D.C., Tarantilis C.D. and Ioannou G., 2012. Solving project scheduling problems with resource constraints via an event list-based evolutionary algorithm. *Expert Systems with Applications, 39: 3983–3994.*

Peteghem V.V. and Vanhoucke M., 2010. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research 201: 409-418.*

Pinedo M., 1995. Scheduling Theory, Algorithms, and Systems. *Prentice-Hall, Englewood Cliffs, NJ.*

Rios-Mercado R.Z. and Bard J. F., 1997. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. *Computers & Operations Research 25: 351–366*

Ruiz R. and Maroto C., 2006. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research 169: 781–800*

Ruiz R., Şerifoğlu F.S. and Urlings T., 2008. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research, 35: 1151 – 1175*

Schwindt C. and Trautmann N., 2000. Batch scheduling in process industries: an application of resource–constrained project scheduling. *OR Spektrum (22): 501–524*

Sprecher A., Kolisch R. and Drexl A., 1995. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research 80: 94-102*

Srikar B.N. and Ghosh S., 1986. A MILP model for the n-job, m-stage flowshop with sequence dependent set-up times. *International Journal of Production Research 24 (6):1459-1474.*

Stafford E.F. and Tseng F.T., 2001. Two models for a family of flowshop sequencing problems. *European Journal of Operational Research 142 (2): 282–293*

Thomas P.R. and Salhi S., 1998. A tabu search approach for the resource constrained project scheduling problem. *Journal of Heuristics 4: 123–139.*

Toklu Y.C., 2002. Application of genetic algorithms to construction scheduling with or without resource constraints. *Canadian Journal of Civil Engineering 29: 421–429.*

Tseng L.-Y. and Chen S.-C., 2006. A hybrid metaheuristic for the resource-constrained project scheduling problem. *European Journal of Operational Research 175: 707–721*

Valls V., Ballestin F. and Quintanilla S., 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research 185: 495-508*

Valls V., Quintanilla M.S. and Ballestin F., 2003. Resource-constrained project scheduling: A critical reordering heuristic. *European Journal of Operational Research 149: 282–301*.

Valls V., Ballestin F. and Quintanilla S., 2005. Justification and RCPSP: A technique that pays. *European Journal of Operational Research 165: 375–386.*

Wuliang P. and Chengen W., 2009. A multimode resource-constrained discrete time-cost tradeoff problem and its genetic algorithm based soution. *International Journal of Project Management 27: 600-609*

Zandieh M., Ghomi S.M.T.F., and Moattar S.M. H., 2005. General models for hybrid flow shop sequencing problems with sequence dependent setup times. *Journal of Faculty of Engineering, 39(4): 1-9*

APPENDIX A

PSEUDO CODE FOR THE GENETIC ALGORITM

Input:

Size $N = 5 \times$ (number of tasks) of population,

Number G=250 of generations,

Initialization:

Generate N precedence feasible activity lists by selecting tasks randomly from the eligible set in serial SGS and save them in the population Pop

For i=1 to G do

Calculate makespan of each activity list in *Pop* by serial SGS and calculate the total setup time of each activity list

First, sort the activity lists in *Pop* in ascending order of makespan and then sort the activity lists in *Pop* in descending order of total setup time

Create empty Pop_1. Size of the Pop_1 is equal to the size of *Pop*

Put the top *20* % of *Pop* into Pop_1

For j=1 to (40 % of the size of the Pop_1) do

Select one parent from the top *20* % of *Pop* randomly and save it to P_1

Select one parent from *Pop* randomly and save it to P_2

Perform one point crossover of P_1 and P_2

Perform setup improvement procedure to the children

For the child 1, if the fitness of the mother is less than the child 1, the order of the tasks that need the machine is tried to be made similiar to the order in the mother's activity list as explained in the 4.1.6. For the child 2, same procedure is applied but mother is replaced with father.

Perform fitness improvement procedure to the children

For the child 1 and child 2, try to improve the fitness by removing and inserting some task in the activity list as explained in the 4.1.6

Perform mutation on two children of one-point crossover:

For k=1 to a randomly selected number between [3, 5] do

Select a task that needs the machine

Put this task into a most setup improvement and precedence feasible position in the activity list.

If there is no setup improvement, put this task into a least setup deteriorated and a precedence feasible position in the activity list with a probability of 0.05

Select a task that needs the machine or do not need the machine

Put this task into randomly selected precedence feasible position if there is a fitness improvement when this task is putted in this randomly selected position

If there is no fitness improvement, put this task into a randomly selected precedence feasible position in the activity list with a probability of 0.05

End for

Put these children into Pop_1

End for

Replace *Pop* with Pop_1

End for

Select the fittest member from *Pop*

APPENDIX B

AVERAGE OF REPLICATIONS FOR PROBLEMS WITH CROSSOVER TYPES

Table 19 *Average result of ten replications for problem j3010_4 with each crossover type*

Problem	Crossover Type	Average of ten replications
j3010_4	One-point	176.8
j3010_4	Two-point	177.4
j3010_4	Three-point	177.6

Table 20 *Average result of ten replications for problem j3045_4 with each crossover type*

Problem	Crossover Type	Average of ten replications
j3045_4	One-point	148
j3045_4	Two-point	148.5
j3045_4	Three-point	148.4

Table 21 *Average result of ten replications for problem j6010_4 with each crossover type*

Problem	Crossover Type	Average of ten replications
j6010_4	One-point	291.9
j6010_4	Two-point	293.9
J6010_4	Three-point	294.6

Table 22 *Average result of ten replications for problem j6045_4 with each crossover type*

Problem	Crossover Type	Average of ten replications
j6045_4	One-point	240.7
j6045_4	Two-point	241.5
j6045_4	Three-point	241.4

APPENDIX C

GENETIC ALGORITHM'S RESULTS AND CPU TIMES OF THE TEN REPLICATIONS FOR EACH PROBLEM.

Table 23 *Results of the genetic algorithm for ten replications of problem j1510_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	103	25.01
2	103	25.28
3	103	24.94
4	103	25.15
5	103	25.32
6	103	24.83
7	103	25.18
8	103	24.71
9	103	24.77
10	103	24.57

Table 24 *Results of the genetic algorithm for ten replications of problem j1510_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	33	19.88
2	33	19.90
3	33	19.45
4	33	19.58
5	33	20.13
6	33	20.25
7	33	20.29
8	33	19.72
9	33	19.25
10	33	19.16

Table 25 *Results of the genetic algorithm for ten replications of problem j1545_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	70	22.00
2	70	23.28
3	70	23.50
4	70	22.94
5	70	23.01
6	70	22.80
7	70	22.54
8	70	22.09
9	70	22.34
10	70	22.36

Table 26 *Results of the genetic algorithm for ten replications of problem j1545_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	59	20.21
2	59	20.28
3	59	20.53
4	59	20.44
5	59	20.75
6	59	20.50
7	59	20.12
8	59	20.18
9	59	20.55
10	59	20.79

Table 27 *Results of the genetic algorithm for ten replications of problem j3010_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	175	88.92
2	175	84.19
3	175	84.25
4	175	84.11
5	175	84.39
6	175	83.39
7	175	84.98
8	175	89.13
9	175	83.75
10	175	83.52

Table 28 *Results of the genetic algorithm for ten replications of problem j3010_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	89	87.56
2	89	87.68
3	89	88.25
4	89	92.05
5	89	87.99
6	89	88.04
7	89	86.85
8	89	89.66
9	89	86.31
10	89	90.32

Table 29 *Results of the genetic algorithm for ten replications of problem j3045_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	148	80.55
2	148	81.38
3	148	80.68
4	148	81.00
5	148	82.27
6	148	80.62
7	148	82.36
8	148	80.48
9	148	81.33
10	148	81.47

Table 30 *Results of the genetic algorithm for ten replications of problem j3045_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	98	76.91
2	95	78.74
3	95	76.91
4	95	77.14
5	98	77.92
6	98	77.60
7	99	77.91
8	98	77.67
9	99	79.37
10	95	77.34

Table 31 *Results of the genetic algorithm for ten replications of problem j6010_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	286	337.58
2	284	338.78
3	282	344.08
4	284	368.40
5	283	337.24
6	284	344.92
7	284	340.40
8	283	333.90
9	286	333.50
10	286	329.34

Table 32 *Results of the genetic algorithm for ten replications of problem j6010_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	137	307.05
2	137	310.56
3	137	308.95
4	137	312.04
5	137	309.90
6	137	309.83
7	137	309.21
8	137	312.84
9	137	314.35
10	137	310.84

Table 33 *Results of the genetic algorithm for ten replications of problem j6045_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	240	300.16
2	238	314.96
3	239	303.83
4	238	304.19
5	240	304.94
6	237	301.40
7	239	294.26
8	240	315.24
9	240	311
10	239	295.95

Table 34 *Results of the genetic algorithm for ten replications of problem j6045_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	162	395.84
2	163	393.87
3	163	397.21
4	163	395.42
5	161	402.72
6	162	396.80
7	164	387.70
8	162	401.73
9	163	381.31
10	163	389.03

Table 35 *Results of the genetic algorithm for ten replications of problem j9010_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	485	890.50
2	481	858.48
3	480	893.04
4	483	875.54
5	484	890.67
6	483	901.34
7	480	871.95
8	482	919.09
9	481	887.15
10	480	885.07

Table 36 *Results of the genetic algorithm for ten replications of problem j9010_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	208	869.63
2	208	754.93
3	206	768.88
4	206	750.60
5	206	748.26
6	206	755.79
7	206	741.06
8	208	744.13
9	208	755.95
10	208	763.21

Table 37 *Results of the genetic algorithm for ten replications of problem j9045_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	392	755.58
2	391	778.45
3	393	799.55
4	393	752.42
5	391	762.22
6	393	835.86
7	391	785.03
8	393	765.14
9	392	816.01
10	392	749.37

Table 38 *Results of the genetic algorithm for ten replications of problem j9045_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	244	865.66
2	240	864.73
3	241	850.07
4	244	859.90
5	241	875.32
6	240	848.36
7	244	887.67
8	240	831.93
9	241	896.99
10	241	869.66

Table 39 *Results of the genetic algorithm for ten replications of problem j12010_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	581	1672.20
2	581	1708.84
3	579	1643.67
4	580	1791.13
5	585	1728.96
6	580	1704.80
7	582	1696.03
8	585	1671.28
9	580	1703.83
10	582	1723.03

Table 40 *Results of the genetic algorithm for ten replications of problem j12010_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	292	1623.20
2	289	1553.12
3	289	1625.70
4	291	1616.01
5	291	1558.11
6	291	1590.81
7	292	1537.14
8	291	1578.79
9	291	1622.89
10	291	1671.41

Table 41 *Results of the genetic algorithm for ten replications of problem j12045_4*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	480	1406.13
2	481	1435.82
3	480	1463.38
4	480	1408.41
5	480	1436.19
6	480	1491.55
7	480	1422.82
8	480	1529.16
9	479	1423.72
10	480	1459.65

Table 42 *Results of the genetic algorithm for ten replications of problem j12045_8*

Replication No	Result of the genetic algorithm	Cpu Time(sec.)
1	288	1598.73
2	285	1502.64
3	286	1460.45
4	285	1569.19
5	285	1557.49
6	288	1529.42
7	285	1512.40
8	291	1506.43
9	286	1476.11
10	286	1519.38

APPENDIX D

A HILL-CLIMBING-LIKE SEARCH ALGORITHM RESULTS AND CPU TIMES OF THE TEN REPLICATIONS FOR EACH PROBLEM

Table 43 *Results of the hill-climbing-like search algorithm for ten replications of problem j1510_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	112	25
2	107	25
3	107	25
4	112	25
5	112	25
6	112	25
7	108	25
8	110	25
9	109	25
10	112	25

Table 44 *Results of the hill-climbing-like search algorithm for ten replications of problem j1510_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	34	19
2	34	19
3	35	19
4	36	19
5	34	19
6	35	19
7	34	19
8	36	19
9	34	19
10	35	19

Table 45 *Results of the hill-climbing-like search algorithm for ten replications of problem j1545_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	70	22
2	74	22
3	70	22
4	70	22
5	70	22
6	72	22
7	70	22
8	74	22
9	78	22
10	72	22

Table 46 *Results of the hill-climbing-like search algorithm for ten replications of problem j1545_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	61	20
2	61	20
3	61	20
4	62	20
5	65	20
6	62	20
7	61	20
8	62	20
9	61	20
10	65	20

Table 47 *Results of the hill-climbing-like search algorithm for ten replications of problem j3010_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	180	84
2	182	84
3	183	84
4	182	84
5	181	84
6	181	84
7	178	84
8	179	84
9	186	84
10	179	84

Table 48 *Results of the hill-climbing-like search algorithm for ten replications of problem j3010_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	93	87
2	97	87
3	94	87
4	90	87
5	100	87
6	98	87
7	90	87
8	95	87
9	91	87
10	94	87

Table 49 *Results of the hill-climbing-like search algorithm for ten replications of problem j3045_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	148	80
2	159	80
3	155	80
4	158	80
5	155	80
6	150	80
7	157	80
8	156	80
9	152	80
10	149	80

Table 50 *Results of the hill-climbing-like search algorithm for ten replications of problem j3045_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	105	77
2	103	77
3	104	77
4	105	77
5	108	77
6	99	77
7	108	77
8	105	77
9	100	77
10	105	77

Table 51 *Results of the hill-climbing-like search algorithm for ten replications of problem j6010_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	302	337
2	297	337
3	298	337
4	293	337
5	300	337
6	303	337
7	295	337
8	301	337
9	302	337
10	297	337

Table 52 *Results of the hill-climbing-like search algorithm for ten replications of problem j6010_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	146	308
2	143	308
3	142	308
4	145	308
5	142	308
6	147	308
7	149	308
8	147	308
9	140	308
10	146	308

Table 53 *Results of the hill-climbing-like search algorithm for ten replications of problem j6045_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	250	304
2	252	304
3	258	304
4	249	304
5	248	304
6	244	304
7	247	304
8	245	304
9	246	304
10	243	304

Table 54 *Results of the hill-climbing-like search algorithm for ten replications of problem j6045_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	175	393
2	175	393
3	170	393
4	175	393
5	170	393
6	178	393
7	173	393
8	173	393
9	178	393
10	173	393

Table 55 *Results of the hill-climbing-like search algorithm for ten replications of problem j9010_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	502	871
2	496	871
3	506	871
4	499	871
5	498	871
6	495	871
7	505	871
8	505	871
9	515	871
10	497	871

Table 56 *Results of the hill-climbing-like search algorithm for ten replications of problem j9010_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	222	763
2	220	763
3	224	763
4	220	763
5	222	763
6	217	763
7	221	763
8	227	763
9	222	763
10	218	763

Table 57 *Results of the hill-climbing-like search algorithm for ten replications of problem j9045_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	409	785
2	414	785
3	416	785
4	414	785
5	416	785
6	406	785
7	413	785
8	413	785
9	412	785
10	407	785

Table 58 *Results of the hill-climbing-like search algorithm for ten replications of problem j9045_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	261	850
2	256	850
3	256	850
4	263	850
5	260	850
6	250	850
7	263	850
8	250	850
9	258	850
10	254	850

Table 59 *Results of the hill-climbing-like search algorithm for ten replications of problem j12010_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	605	1708
2	615	1708
3	608	1708
4	608	1708
5	613	1708
6	613	1708
7	609	1708
8	620	1708
9	611	1708
10	617	1708

Table 60 *Results of the hill-climbing-like search algorithm for ten replications of problem j12010_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	319	1529
2	324	1529
3	303	1529
4	312	1529
5	307	1529
6	298	1529
7	310	1529
8	310	1529
9	310	1529
10	314	1529

Table 61 *Results of the hill-climbing-like search algorithm for ten replications of problem j12045_4*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	503	1422
2	502	1422
3	504	1422
4	502	1422
5	499	1422
6	502	1422
7	499	1422
8	496	1422
9	503	1422
10	498	1422

Table 62 *Results of the hill-climbing-like search algorithm for ten replications of problem j12045_8*

Replication No	Result of the search algorithm	Cpu Time(sec.)
1	306	1476
2	303	1476
3	311	1476
4	308	1476
5	306	1476
6	311	1476
7	307	1476
8	314	1476
9	311	1476
10	308	1476