DISSERTATION

Integration Of Genetic Algorithms For Knowledge-based Multi-Objective Scheduling Problems in the DéjàVu Class Library

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften unter der Leitung von

a.Prof.Dr.Jürgen Dorn
Institut für Informationsysteme

eingereicht an der Technischen Universität Wien
Technisch-Naturwissenschaftliche und Infomatik Fakultät

von

Dipl.-Ing. Maamar Saib
Matrikelnummer: 952 75 12
Linzer Strasse 429/4012, A-1140 Wien

Wien, am Oktober 2002

بسم الله الرحمن الرحيم

Abstract

Real-world scheduling problems are confronted with conflicting constraints and objective functions. Scheduling problems are known as *NP-Hard problems*, i.e. there is no algorithm that solves them in polynomial time. In the artificial intelligence community many approaches are used to solve these problems. In the last three decades modern approaches from artificial intelligence were used. These approaches are known as meta-heuristic search methods, such as simulated annealing, tabu search, evolutionary algorithms, random search, and neural networks. In this thesis we use evolutionary algorithms to try to solve multi-objective production scheduling problems in high-grade steel making. We focus on the use of genetic algorithms, which are inspired from the Darwinian theory of the evolution in nature and are some of the most used algorithms in the evolutionary computation community. Genetic algorithms are integrated in a framework called *DéjàVu* class library, which is specially designed for the production scheduling problems in high-grade steel making.

DéjàVu class library is based on object oriented paradigm, where other improvement methods like tabu search are implemented. In this thesis genetic algorithms try to improve the multi-objective scheduling problem and compare the result with tabu search method. As application a steel demo from the steel industry is used, which was inspired by real-world steel making scheduling problems.

The Unified Modeling Language (UML), a modeling language for software design, is used to design the integration of genetic algorithms in the DéjàVu Class library. Numerous representations of a schedule in a chromosome and the corresponding operators are discussed. Several genetic algorithms parameters like population size, crossover operators, mutation operators, crossover and mutation probability are changed to see the influence of these operators on the performance of genetic algorithms.

In this thesis we extended the *DéjàVu* Class library to integrate genetic algorithms and to do experiments to see how effective genetic algorithms are in solving production scheduling problems in high-grade steel making. In the experiments we used two kinds of genetic algorithms, the *simple genetic algorithm* and the *steady state genetic algorithm*. We compared the results with the tabu search method and we proved that if we use the appropriate parameters for genetic algorithms we get better results then when using the tabu search method.

Keywords

Genetic Algorithms, Evolutionary Algorithms, Real World Scheduling Problems, Multi-Objective Scheduling, Constraint Satisfaction Problems, Tabu Search, Crossover Operators for scheduling, Mutation Operators for Scheduling, Object Oriented Analysis and Design, Unified Modeling language(UML).

Kurzfassung der Dissertation

Praktische Anwendungen des Schedulingproblems sind oft mit widersprüchlichen Bedingungen und Zielfunktionen konfrontiert. Fragestellungen werden auch als NP-schweren Probleme bezeichnet, dh. es gibt keinen Algorithmus der dieses Problem in polynomialer Zeit löst. Im Bereich der künstlichen Intelligenz gibt es viele Ansätze dieses Problem zu lösen. In den letzten drei Jahrzehnten wurden viele Ansätze der künstlichen Intelligenz verwendet. Diese Ansätze werden unter dem Begriff metaheuristischen suche zusammengefasst, dazu gehören Simulated annealing, tabu search, evolutionäre Algorithmen, Zufallsuche und Neurale Netze. In dieser Dissertation verwenden wir evolutionäre Algorithmen um multikriterielle scheduling Probleme im Stahlerzeugungsprozess zu lösen. Wir verwenden genetische Algorithmen, die aus der Darwinschen Evolutionstheorie entstanden sind und weit verbreitet im Bereich der evolutionären Algorithmen sind. Die genetische Algorithmen werden in einer Klassenbibliothek mit dem Namen ”*Déjà Vu* class library” integriert, welche für Schedulingprobleme in der Stahlerzeugung konzipiert wurde. Die ”*Déjà Vu* class library” ist objekt-orientiert ausgerichtet und beinhaltet Methoden wie tabu-search. In dieser Dissertation versuchen genetische Algorithmen multikriterielle scheduling Probleme zu lösen und die Ergebnisse mit tabu-search zu vergleichen. Als Anwendung wird eine ”steel-demo” der Stahlindustrie verwendet, die den realen Bedingungen der Stahlerzeugung entspricht. Zur Integration der genetischen Algorithmen in die ”*Déjà Vu* class library” wird die ”Unified Modeling Language” (UML) verwendet. Mehrere Repräsentationen eines ”schedules” in einem Chromosome und dessen Operatoren werden diskutiert. Einige Parameter der genetischen Algorithmen wie zum Beispiel Grösse der Population, ”crossover”-, Mutations- Operatoren, Crossover- und Mutationswahrscheinlichkeit wurden modifiziert, um den Einfluss dieser Operatoren auf die Qualität der genetischen Operatoren zu untersuchen. In dieser Arbeit wurde die "*DéjàVu* class library" erweitert, um genetische Algorithmen zu integrieren und zu untersuchen, wie effektiv genetische Algorithmen verwendet werden kø"nnen, um "scheduling Probleme" im Stahlerzeugungsprozess zu lösen. In den Experimenten verwendeten wir zwei Arten von genetischen Algorithmen: "simple genetic algorithm" und "steady state genetic algorithm". Wir verglichen die Ergebnisse mit der "tabu search " Methode und bewiesen, dass unter Zuhilfenahme der passenden Parameter für die genetischen Algorithmen bessere Resultate, als mit "tabu search" zu erreichen sind.

Acknowledgments

I thank all persons, who aided my research specially a.Prof.Dr.Jürgen Dorn my adviser, Prof. Georg Gottlob, Dr.Dipl.Ing. Marion Girsh, a.Prof.Dr. Dipl. Ing. Wolfgang Slany. I thank my family who supported me and my friends like in Vienna Ahmed Belbachir Nabil, Ait-Tahar Rachid, and my friends in Algeria.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter an overview of the problem which is being studied will be presented as well as the methods which are used to try to solve it. The problem is a multi-criteria scheduling problem, which is an *NP-Hard problem* [164]. For the past three decades this problem was solved with two kind of methods. Methods from operational research field, where the search for a solution is deterministic, and methods from artificial intelligence field, where these methods are non-deterministic. The deterministic methods from operational research field try to find an exact solution for the scheduling problems, and the non-deterministic methods try to find an approximate solution. This thesis tries to solve the scheduling problem with a class of methods from artificial intelligence field called evolutionary computation methods. These methods are meta-heuristic search methods [170] which begin with several initial solutions and try to improve these solutions with an evolutionary computation system. In this chapter an overview is given of both the general and the particular scheduling problems. Then an overview of the evolutionary algorithms and the difference between them is provided.

1.1 Scheduling Problems

Scheduling problems are well known as combinatorial problems which are defined as follows:

Assume we have a set of machines $M_i(i = 1, .., m)$ and a set of jobs $J_j(j = 1, ..., n)$. The schedule is an allocation of the operations of each job J_j on the machines M_i respecting some constraints like those of temporal precedence, constraints on resources and any others while optimizing a set of objective functions like makespan, or other objective functions. Several scheduling problems exist, which are described in [30] and [5]. Some of them are presented as follows:

1. Job shop schedule: The operations of jobs are scheduled in different orders on machines respecting some constraints like:
   -Each machine can process only one job at a time.
   -The sequence of operations for each job is predefined
   -Two operations of the same job cannot be processed at the same time

2. Flow Shop Schedule: It is a special case of the job shop. In this case all the operations of a job are scheduled in the same order on the machines. The schedule is considered as a permutation of all jobs to be scheduled.

3. Open shop schedule: This is the same as the flow shop with the exception that there are no precedence relations between the operations of a job. The operations of a job can be processed in any order, as long as no two operations are processed on different machines simultaneously.

4. Mixed job schedule: This is a combination of a job shop and open shop schedule.

5. Machine Scheduling Problem: In this case n jobs j_1,... j_n are scheduled on a single machine [100]. Several objective functions can be optimized, like minimizing the squared deviations from the mean completion time or flow time. The constraints could be if we have two jobs i and j, i must be processed before j on a machine.

Several approaches were applied to solve scheduling problems, here is an overview of these approaches:

1. Mathematic methods from the operations research community: Optimization techniques such as linear programming and dynamic programming, were not successful when they were applied to scheduling problems with a large number of machines and jobs or for real world scheduling problems as in the manufacturing industry.

2. Heuristic methods from combinatorial optimization theory: In this approach the heuristics produced good solutions but not guaranteed optimal solutions [114]. One of the most used methods in this approach is branch and bound heuristic, which is not used in real world scheduling problems because of a large number of constraints and optimization functions.

3. Heuristics search methods from the artificial intelligence community: In the artificial intelligence community, scheduling problems are considered as constraints satisfaction problems. These are solved using some heuristic search methods, like simulated annealing, tabu search, evolutionary algorithms. Section 1.2 will present greater overview of these search methods.

4. Fuzzy sets and fuzzy logic:

   In Slany's [181] dissertation, he used fuzzy sets and fuzzy logic to represent constraints, which are imprecisely defined.

5. Methods that mimic natural process:

   In this case many methods inspired by nature were used such as simulated annealing, evolutionary algorithms, neural network, and case-based reasoning.

6. Expert Systems Approach:

   In Dorn [65] expert systems were presented to solve real world scheduling problems in the steel making industry. One of the expert systems is the *REPLAN system*; it is explained by Dorn [65] as follows:

   "The REPLAN system supports the Human experts in construction a schedule for high-grade steel making at Böler in Kapfenberg, Austria. The plant consists of four production lines each starting with an electric arc furnace. The molten steel is poured into ladles, refined in special aggregates, and finally cast either by cranes into ingots on a horizontal continuous caster into slabs. The greatest problem is the variety of metallurgical grades... Besides chemical constraints, spatial and temporal constraints have to be regarded "

   The approach to solve the steel scheduling problem is described in Dorn [65] as follows:

   "The scheduling system first analyses all orders for one week and determines what are the problems or bottlenecks in this week. Then each order gets a value that describes

> the difficulty to schedule this order in the actual week. The important orders and those that were classified very difficult will be scheduled first.... For some orders this can result in constraint violations. Finally, the achieved schedule is analyzed for week points and is iteratively repaired..."
>
> A fuzzy sets were used to represent the importance and difficulty of jobs and the degree of constraint satisfaction [59] in order to compare different schedule created by the system. Evaluation functions were defined by combining the fuzzy values of single constraints.
>
> Scheduling in the steel making industry is presented as a knowledge based system, where the knowledge of the schedule are like job, order, resource,.., are modeled with an object oriented design paradigm.

This thesis tries to solve scheduling problems with evolutionary algorithms, specifically with genetic algorithms. There are two reasons why genetic algorithms are used. The first reason is that deterministic methods do not guaranty an optimal solution or "good" solution in polynomial time whereas the genetic algorithms can find "good" solutions in a reasonable time. The second reason is that we want to see how well genetic algorithms compare to other search algorithms from artificial intelligence. We will compare genetic algorithms to the tabu search method and also compare genetic algorithms to each other. Through experiments we will see how parameters of genetic algorithms influence their performance. In the next section an overview of the evolutionary algorithms is presented, which have became extremely useful in solving many kinds of problems.

1.2 Overview of the Modern Search Methods from Artificial Intelligence

1.2.1 Simulated Annealing

Simulated Annealing [170] is one of the modern search methods which was inspired by nature. It was first used in statistical physics to simulate the cooling of material in a heat bath. If solid material is heated past its melting point and then cooled back into a solid state, the structural properties of the cooled solid depend on the rate of the cooling. The probability of an increase in energy of magnitude (σE) is as follows:

$prob(\sigma E) = exp(-\frac{\sigma E}{kt})$ Where t is the temperature, k is a physical constant known as

Boltzmann's constant. The algorithm of simulated annealing is given in figure 1.1:

```
1. Select an initial solution $s_0$
2. Select an initial temperature $t_0 < 0$
3. Select a temperature reduction function $\alpha$
4. Repeat
   (a) Repeat
       i. Randomly select $s \in N(s_0)$
       ii. $\sigma = f(s) - f(s_0)$
       iii. if $\sigma < 0$
       iv. then $s_0 = s$
       v. else
       vi. generate random $x$ uniformly in the range (0, 1);
       vii. if $x < \exp(-\frac{\sigma}{t})$ then $s_0 = s$;
       Until iteration_acount = nrep
   (b) Set $t = \alpha(t)$;
   Until stoppingcondition = true.
5. $s_0$ is the approximation to the optimal solution.
```

Figure 1.1: Simulated Annealing Algorithm

1.2.2 Tabu Search

Tabu search [170] is a search method based on the use of flexible memory. The memory structure of tabu search operate by reference to four principal dimensions, consisting of recency, frequency, quality, and influence. The algorithm of tabu search is presented below:

- *Step 1: INITIALIZATION*
  Begin with the same initialization used by neighbourhood search and with the history record H empty.

- *Step 2: CHOICE AND TERMINATION*
  Determin $Candidate\_N(x^{now})$ as a subset of $N(H, x^{now})$. Select x^{next} from $Candidate\_N(x^{now})$ to minimize $c(H, x)$ over this set. (x^{next} is called a highest evaluation element of $Candidate\_N(x^{now})$. Terminate by a chosen iteration cut-off rule.

- *Step 3: UPDATE*
  Perform the update for the neighborhood search method and additionally update the history record H.

Tabu search method is considered as neighborhood search method.

1.2.3 Evolutionary Algorithms

Evolutionary algorithms are one of the machine learning methods. They use the principle of adaptation in nature. In Bäck, [7] evolutionary algorithms are examples of unsupervised learning techniques; i.e. inductive learning by observation and discovery. Bäck [7] gives two reasons why evolutionary algorithms are classified as machine learning methods, these reasons are described as follows:

1. "No teacher presents examples, counterexamples or even knowledge to the learning system. Instead, the algorithm generates examples on its own."

2. "The creation of new examples (search points) by the algorithm is an inductive guess on the basis of existing knowledge. If the guess proves its worth, it is kept in the knowledge base (the population), otherwise it is discarded by means of selection"

"Evolutionary algorithms are related to artificial intelligence as an example of artificial intelligence programs" [7]. evolutionary algorithms or evolutionary Programs that are described by Michalewicz [148] are general search algorithms based on the idea of the evolution in nature.

We distinguish between several kinds of evolutionary algorithms as follows:

1. Genetic Algorithms: They simulate Darwinian evolutionary process, they were first developed by Holland in the seventies [108] and became famous by Goldberg in his book [93] and subsequently studied by [53], [120] and others. In this thesis genetic

algorithms will be used as search methods to solve the scheduling problems; more details will be seen in chapter 3

2. Evolutionary Programming: It was created by Lawrence Fogel in 1963 [84], this method searches through a space of small finite state machines.

3. Evolutionary Strategies: They were created in 1973 by Ingo Rechenberg [169] and [179]. They work on a smaller populations.

4. Genetic Programming: It was created by John Koza [120] in 1992 and was inspired by genetic algorithms. The genetic programming paradigm was created to respond to the question posed by Koza [120] "How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it". The structure used by Koza [120]was computer programs.

The different steps of the evolutionary algorithms are presented in the next subsections.

Initialization

Initial solutions are generated randomly or using some heuristics. Sometimes the performance of evolutionary algorithms depends on the initial solution. The initial solution can also be created with local search methods like hill-climbing.

Fitness assignment

Each phenotype corresponding to a genotype in evolutionary algorithms is evaluated with an evaluation function and according to this evaluation function a fitness function is assigned to each genotype.

Selection

The fitter solutions are selected from the population in each generation with a selection method. The selected solutions are candidate to reproduce a new generation.

Reproduction

In this stage of the evolutionary algorithms the selected parents produce children with recombination and mutation operators to create a new generation. The new generation is made up of children, who inherit characteristics from their parents. They differ from their parents but they have preserved some of their characteristics. [93]

Termination

The termination of the evolutionary algorithms depends on some criteria. These are usually solution quality, number of generation, or convergence of the evolutionary algorithms.

1.3 Conclusion

There is no deterministic method to solve scheduling problems in polynomial time. Evolutionary algorithms and especially genetic algorithms are stochastic search methods, which allow us to solve scheduling problems more expeditiously as well as improving the solution to create the best schedule. Often research trying to solve scheduling problems with genetic algorithms or other evolutionary algorithms does not use real-world scheduling problems. This thesis will use real world problems as framework to create solutions for scheduling problems in the artificial intelligence field.

1.4 Structure of the Thesis

Section 2 will present scheduling problems and especially the steel demo scheduling problem.
Section 3 will present genetic algorithms in general.
Section 4 will present a review of representations and operators for scheduling problems.
Section 5 will explain the design and integration of genetic algorithms in the existing *DéjàVu class library.*
In section 6 experiments are presented which compare the results of genetic algorithms and the tabu search method as well as comparing the results between several kinds of genetic algorithms and look at how genetic operators influence the performance of the solution.
Finally, section 7 will provide a general conclusion and the open topics.

Chapter 2

Production Scheduling

2.1 Introduction

In this chapter a production scheduling problem in steel making process is described, which is a real world scheduling problem. This scheduling problem is inspired from the steel making plant at Böler Uddeholm in Kapfenberg (Austria) [72]

2.2 Scheduling of High-Grade Steel Process

Many publications were made by Dorn et al. to solve the scheduling of *High-Grade Steel Process*. [59] [69] [65] [61] [73] [60] [62] [72] [70] [68] [67] [66] [74] [64] [63]. In the following a representation of the problem is done.

2.2.1 Presentation of the problem

As described in [72] a general presentation of the problem being solved is presented as follows: Böler Uddeholm is one of the most important European producers of high-grade steel. High-grade steel is crude steel refined with alloying metals like manganese, tungsten, chromium, or others. These alloying metals have desired effects like compression strength, impact strength, and many more. A production planning system compounds orders to jobs and chooses required process plans. Furthermore, the system determines weeks when these jobs shall be processed by regarding the capacity of the steel making shop. It works out for each week two sets of jobs to be processed on both production lines. Usually the first shift in the steel making plants starts Sunday evening and the last shift ends Saturday. Sometimes fixed sequences for two or three jobs are given to facilitate the scheduling in the subsequent plant. The task of the scheduler is to find a possible sequence for all jobs with violating as few compatibility constraints between jobs as possible and to allocate resources over time without violating temporal and capacity constraints. The result of this scheduling process may be that some orders are rejected and shifted to the next week. To reduce the number of rejected orders, general rules that explain what combinations of orders may be produced in one week are already regarded by the production planning system. The steel making plant produces slabs or ingots of a certain quality for different subsequent plants like the forge, the rolling mill, or the foundry. The destination is important for the scheduling process, because the working hours of these plants must be considered. Sometimes products are stored for several days in intermediate stock yards, because the next plant cannot process the jobs in the same sequence as the steel making plant. The different sequencing criteria of jobs cause considerable costs for the company. Moreover, since the steel cools down it must be warmed up again in the next plant. To reduce costs and to improve the quality, some orders have due dates.

The steel Making Process

Pig iron produced in blast furnaces contains usually more than 4% carbon and is therefore brittle. To get a deformable products, steel is produced by reducing carbon in pig iron down to 0.2%. For many high-tech products the quality of steel must be even higher and reductions of different chemical elements that are still in the steel must be achieved. If the steel would be delivered immediately from the blast furnace this reduction process could be made in LD-converts. Since in Kapfenberg no blast furnace exists the crude steel and scrap iron are Matelda in electrical are furnaces. To reduce costs scrap iron with high percentages in desired alloying elements is used.

The steel making shop in Kapfenberg consists of three main production lines that share some aggregates. For every steel quality there is a process plant that describes which operations must be performed on which aggregates to produce the quality. These operations and sequences must be only replanned when a failure occurs.

The steel making process starts with the charge of crude steel and scrape iron in one of the three electric arc furnaces (EAF). The filling of one furnace is called a heat and it already contains the main alloying elements. The furnaces have different capacities from $17t$ to $55t$. the duration of melting one heat depends on the size and the ingredients but also external causes. Since the furnaces consume much electric energy, they are sometimes switched off due to voltage peaks. Up to five hours can be required for melting, but usually two to two and half hour are enough. A fixed set-up and a maintenance interval of twenty minutes are included in this interval. The liquid steel is poured into ladles that are transported by a crane to a ladle furnace (LF). If the preceding heat has a long processing time in the ladle furnace, the current heat must wait. This slack time may not exceed two hours. The next step is a heat treatment in the ladle furnace where where the fine alloying takes place and whose duration is usually about the same as melting. Later a special treatment may be performed in the vacuum oxygen decarburation (VOD) unit or on the vacuum decarburation (VD) unit. The VD-unit can be converted into a VOD-unit. This conversion takes about three to four hours.

The latest step in the steel making shop is the processing of the steel in a horizontal continuous caster (HCC) to form slabs or the casting of it into molds to form ingots. The teeming rate for the HCC is about $50t/h$. If the casting format must be altered, a set-up time must be considered, too. For casting ingots, space in one of the four teeming bays (TB), where ingots can solidify in the moulds, is required. Normally the solidification time for ingots in hours is half as much as the weight of the ingots in tons. Thus an $52t$ ingot

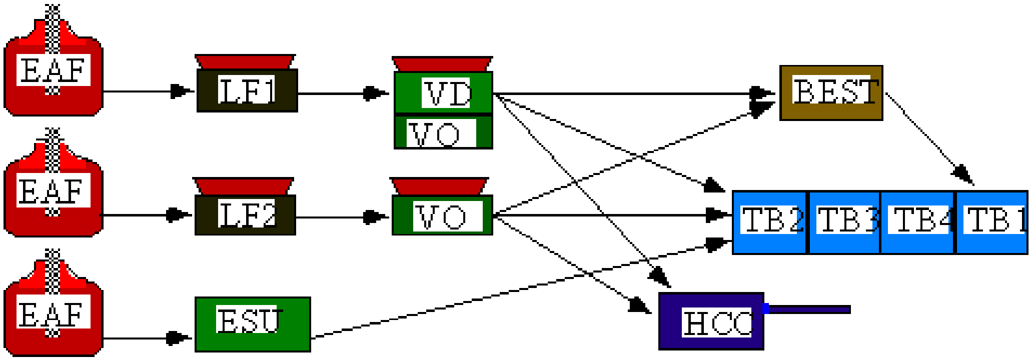


Figure 2.1: Aggregates in the Steel Making Shop

needs one day and for an ingot of 1t half an hour is sufficiant.

The Böhler-Electro-Slag-Topping (BEST)- technology is a special casting technology for big ingots. The ingots are treated additionally in the BEST-unit and for them only one place in the teeming bays (TB 1) exists. The ESU-unit is used for some extra-clean steel qualities. All aggregates and the routings for heats are shown in figure 7.2. Since the third production line EAF5 and ESU-unit is used only seldom it is not regarded further in scheduling.

Constraints in Scheduling

- Compatibility Constraints: As described in [72], the main problem in scheduling is that residuals of one heat in the electric arc furnace may pollute the next heat. The engineers use as a rule of thumb that 3 % of a chemical element in a heat remain in the wall of the electric arc furnace and 3 % of the difference of the elements in two consecutive heats will be assimilated by the second heat. Of course, the 3 % are always on the safe side and in some cases the expert relaxes this factor. Although the rule is effective for 42 chemical elements, usually only eight main elements are considered. These elements are nickel (Ni), Chromium (Cr), manganese (Mn), molybdenum (Mo), tungsten (W), cobalt (Co), Vanadium (V), and iron (Fe). The amount of the other 36 elements is usually very small and restrictions are no scheduling problem. Since not all elements react uniformly, exceptions exist that must be handled separately.

- Temporal Constraints: In principle, the cast steel can be stored in intermediate stocks before it is further treated in one of the subsequent plants. However, then the steel must be reheated which causes considerable costs. For some qualities it is also better if does not cool down. Consequently appointments are made between steel shop and subsequent plants that must be hold within a tolerance of more or less 2 hours. The average number of jobs with such due dates is about 10 %. Of course, these constraints are not really hard since they may be relaxed through negotiations with the subsequent plant. However, it is desirable to hold these due dates to reduce the time needed for negotiations. Furthermore, also these subsequent plants have to regard also compatibility constraints that are different to those of the steel shop.

- Capacity and Related Constraints: About 70 % of the jobs are into ingots and the remaining 30 % are processed on the continuos caster. For both casting technologies some capacity constraints have to be regarded:
  The continuous caster can be supplied with steel from both production lines.

2.2.2 Evaluation of Schedule

In [70] it is described how to evaluate a schedule. A schedule is evaluated in term of the degree of satisfaction of the applicable constraints. Schedules with high satisfaction scores are good schedules. A schedule for which the degree if satisfaction if any constraint is below the critical threshold for that constraint, is not feasible. The evaluation of a schedule S is as follows:

$$f(S) = \frac{1}{n} * \sum_{i=1}^{n} J_i \in S \wedge importance(J_i) + \sum_{j=1}^{m} C_j \in S \wedge satisfaction(C_j) * weight(type(C_j)) \tag{2.2.1}$$

Where $importance(J_i)$ is a fuzzy value explaining the importance of a job J_i. n is the number of existing jobs. The degree satisfaction of a constraint is given by the function satisfaction C_j. Since not all constraints have the same importance for the schedule evaluation, the satisfaction is weighted by a type-specific factor. The function $f(S)$ is used as a fitness function for the genetic algorithms.

2.3 Presentation of the Steel Demo Application

In [71] the Steeldemo was made as application. The steel demo application can be used to construct schedules for a hypothetical steel making plant. The first and most important resource in the plant is an electric arc furnace where the steel is smelt together with scrap iron and alloying elements. We assume a constant duration of 120 minutes for this process. The content of a furnace, the so called heat has up to 50 tons weight. One of the most important constraints on the schedule construction process is the compatibility of steel grades sequenced on this furnace. Chemical elements such as nickel, chromium and others will remain partly in the furnace. The next heat may take over some of these elements. If it is prescribed for the subsequent heat to have only very few parts of this element, the steel may be spoiled by the infiltration from the prior heat. Thus the first objective of the scheduler is to find a good sequence on the furnace. If human experts consider these compatibility constraints, they first formulate a very strong constraint to relax if they do not find a possible sequence. These constraints may be relaxed because there are still possibilities to manipulate the chemical reactions during the actual process. After this first operation the liquid steel is poured into a ladle. With this ladle the steel is transported to different steel aggregates in the secondary metallurgy. The individual resources such as ladle furnaces, a vacuum decarburation unit and others have not to be considered here because they do not pose constraints on the schedule construction process. The duration of the whole secondary metallurgy process depends on the kind of steel quality (i.e. the steel grade) to be produced and is about two to three times longer than the smelting process. In the simplified model of this demo we assume that the time is two times the furnace time plus for each alloying element 20 minutes. An alloying element is a chemical element such as nickel that is added in a considerable amount to give the steel certain characteristics. To enable continuous operation on the furnace, six ladles are available. Since the compatibility problem holds also for the ladles, the scheduler will typically create "classified" ladles where each ladle is specialized for a certain alloying element. Furthermore, the energy consumption on the ladles should be minimized. Since a ladle must be hot if the steel is poured into, one has to use as few ladles as possible and the duration between ladle usages should be minimized without violating other important constraints. Finally, the steel is cast either with a continuous caster unit into slabs or with a casting crane into ingots of certain size. Although the casting on the continuous caster is most economical, only a restricted number of heats may be produced here due to technological constraints that are not under the scheduler's control. If the heat is cast into ingots, these ingots must be prepared and set up by a work force in the teeming bay. After solidification, the ingots must be stripped off by the workers. The durations of the solidification as well as set up and strip off are dependent on the size of the ingots. If the heat is cast into many small ingots the work load is greater than for one or few bigger ingots. Thus it is a further objective to distribute heats to be cast into many small ingots over the whole scheduling period. The special characteristic of steel and the requirement of a prescribed temperature demands that the smelting process, the ladle process and casting process must be immediately after each other. Only very short breaks are allowed. The plant operates in a three-shift mode from Monday 6 am until Saturday 2 pm thus allowing between 50 and 60 heats to be produced each week. Which heats shall be produced are specified in a list of orders.

Such orders may contain additional constraints: There may be a due date demanding the heat to be finished at a certain point of time which has to be hold within a tolerance of two hours. Such orders must be delivered in time. A too late as well as a too early finish is not desired because the cast steel should be still warm for subsequent processing. The preferred start time is a softer time constraint used here to specify for difficult steel grades when they are to be started. These temporal constraints are also relaxable if otherwise no good sequence in regard of the compatibility constraint can be found. Moreover, there may be constraints that describe that two orders shall be produced together. These so called double cast orders are applied if so much steel of a certain grade is ordered that can not be produced in one heat but is also not enough for two legal heats. Therefore the furnace is first filled totally and after smelting, only one part of the smelt heat is poured into the ladle. Then the furnace is refilled to produce the second heat. The steel demo schedulers allows to produce randomly list of orders with the described characteristics. In a preference dialog the number of orders and variability of these orders may be constrained. For example, the number of orders is determined by a random number. However, this range of the random numbers can be constrained. The default setting is to generate between 50 to 60 orders. These values can be changed to some reasonable values. Further it can be restricted how many orders with due dates and preferred start times shall be generated for each list. The number of double cast orders and orders to be produced on the caster are also determined randomly. The diversity of steel grades generated is also under control. So first there is a possibility to constrain how many different steel grades can be produced. If the grade-order ratio (some value between 0 and 1) is small only few different steel grades are generated and the compatibility problem will be not so difficult. If the ratio is set to one, each order will have a different steel grade. However, this will not necessarily result in a difficult problem, because the grades could be similar. The diversity of the steel grades is determined by their chemical content. In the demo scheduler seven chemical elements (nickel, chromium, maganes, vanadium, carbon, nitrogen and iron) are considered. For each element lower and upper limits may be specified to control the diversity of products to be scheduled. The algorithm will first decide how many alloying elements a new steel grade shall contain. The possible alloying elements are nickel, chromium, manganese, and vanadium. At least three are selected randomly. To be an alloying element the content must be higher than 2 elements (especially carbon and nitrogen) lower limits are applied. Finally, for each order it is determined whether slabs on the continuous caster or ingots in the teeming bay are produced. Furthermore, the size and the amount of these products is determined. This decision is independent of the steel grade.

-EAF(Electrical Arc Furnace): In the EAF resource the steel is smelt down, this resource is a non-sharable resource where any two operations(allocations on the resource) may not overlap temporally.

-Ladles(Ladle 1 ..Ladle 6): This is a resource group containing six ladles (non-sharable resources) that are used to transport the liquid steel in the secondary metallurgy until the steel is cast into ingots in the teeming bay or slabs on the continuous caster.

-Continuous Caster(CC): This resource is a non-sharable resource, It should work without interruptions (in sequences) to produce most economically. If the caster stops a longer setup is required. Compatibility constraints are very hard so that usually only equal steel grades may be cast in sequence.

-Teeming Bay(TB): The TB is a sharable resource where the number of blocks in the teeming bay is represented by a curve. If the steel is cast into ingots space for the solidification process is needed in the teeming bay. The TB stores how many ingots solidify at a given point of time in the bay. This resource is sharable because more than one job may use this resource at a time.

-Worker: This resource shows the effort of the workers for building up and strip off the blocks in the teeming bay.

A minimum of fifty jobs and maximum of sixty jobs should be scheduled on these resources, the schedule of these jobs should satisfied some constraints. The main criteria that constraints the sequence of jobs is compatibility constraints between jobs. There are hard constraints and soft constraints; for example Due dates are considered as soft constraints. To evaluate a given schedule the satisfaction of all constraints and objectives are described. Different kind of constraint violation get different weights. In the steel making application we have the following constraints with their weight and threshold: From these constraints there exist soft and hard constraints. Soft constraints can be violated but hard constraints must not be violated, this means to have a feasible schedule hard constraints must be satisfied.

Evaluation Criteria	Weight	Threshold
Chemical compatibility	0.2	0.3
Idle time	0.05	0.0
Tardiness	0.3	0.1
Preferred start time	0.1	0.0
Minimal resources	0.1	0.0
Balanced load	0.05	0.0
makespan	0.2	0.0

Table 2.1: Steel Demo Scheduling Constraints

2.4 Conclusion

In this chapter a general description of steel making plant scheduling was presented and an application, Steeldemo scheduler, which create benchmarks to make experiments on it. The scheduling problem in the High-Grade steel making is a NP-hard problem [164] which could be not solved by traditional methods from operational research. The approach used in this thesis is evolutionary algorithms that are robust methods to find ”good” solution in a polynomial time.

Chapter 3

Genetic Algorithms: An Overview

3.1 Introduction

This chapter provides an overview of the genetic algorithms and explores their theoretical aspect. First, genetic algorithms and their applications will be described, then theoretical description of genetic algorithms is provided as well as how a solution in a genetic algorithms should be represented. Different selection methods, which provide the ability to select the genomes from the population are given and different scaling methods are explained in details. The most important process in genetic algorithms are their operators, namely crossover and mutation operators which are explained in detail in this chapter.

3.2 Genetic Algorithms

Genetic algorithms are one of the most important and the most used techniques in the evolutionary algorithms paradigm. Developed by John Holland [108], genetic algorithms are inspired from the evolutionary process, which exists in nature.

The basic idea of genetic algorithms is that they mimic the evolution to solve optimization problems which have not yet been solved through deterministic methods. These optimization problems are known *NP-Hard problems* [164], such as traveling salesman problem, scheduling problems, Coloring problem,..etc.

Genetic algorithms have several characteristics which are as follows:

1. Representation of solutions
2. Recombination
3. Selection
4. Reproduction
5. Fitness function.

The characteristics above are described below.
Classical genetic algorithms as described by Holland [108] use bit string representation; each string has a fixed length.
The next sections present a theoretical approach of genetic algorithms followed by a discussion of the characteristics of genetic algorithms.

3.3 Theoretical Aspect of Genetic Algorithm

The theoretical aspect of genetic algorithms helps us to understand the mechanisms of genetic algorithms and to answer the questions of why they work and how they work. The widely used theoretical explanation of genetic algorithms was developed by Holland [108] and Goldberg [93]. Some explanation of the theoretical aspect is as follows:
There exist some strings which have similar substrings, this similarity is described by schemata (set of schema). Schemata are defined to be strings of length l over the alphabet $0, 1, *$
Example:
The schema $H = 1 * 01*$ represents a set of the following bit strings
10010
10011
11010
11011
Schemata have two measures Holland [108]:
order $o(H)$ of a schema: number of fixed positions of H.

$H = 0 * *1*$
$o(H) = 2$
defining length $\Delta(H)$ of a schema H, which is a distance between the first and last specific string position.
$H = 0 * *1*$
$\Delta(H) = 4 - 1 = 3$
The influence of selection, crossover, and mutation operators on schemata is described by the following schema theorem:

$$m(H, t+1) \geq m(H, t).f(H)/[1 - Pc.(H)/(l-1)] \tag{3.3.1}$$

The equation 3.3.1 means that short, low-order, above average schemata, so called building blocks, receive exponentially increasing trials in the following generations [93].

3.4 Representation

Representation is the basis of genetic algorithms, because genetic algorithms work on chromosomes.
Simple genetic algorithms, as described by Holland [108], use bit string representation, i.e. each chromosome represents a set of $0, 1$ and each chromosome has a length l.
Example:
Chromosome: 1 0 1 0 0 0 1 0
In the last three decades several new genetic algorithms have been developed, which have different representations. An overview of these representations is as follows:

1. Order-based representation:
   Bean [15], where each element of the chromosome is an integer. This kind of representation was used for sequencing problems like Traveling Salesman Problem (TSP) or scheduling problems.
   Example:
   Chromosome: 1 2 3 4 5 6 7 8

2. Tree structures:

   This kind of representation was created by Koza [120], where each node of a tree represents a program. From this structure new paradigm of genetic algorithms was created called *genetic programming* paradigm.

3. Real representation:Alden [2].

   Where each element of a chromosome represents a real number.

   Chromosome:

   $CH = (x_1, .., x_i, .., x_n)$, where x_i is real number.

4. Matrix Representation [87] Where each element represents a binary (0, 1)
   $$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

3.5 Selection Methods

1. Roulette-Wheel Selection: Goldberg [93]

   This method is also known as proportional selection method Holland [108]. This method of selection works as follows:

   - Calculate the fitness value f_i for each chromosome i in the population (i=1 ,.., pop_size).
   - Find the total fitness of the population
     $F = \sum_{i=1}^{pop\_size} f_i$
   - Calculate the probability of a selection p_i for each chromosome i in the population (i=1 ,.., pop_size):
     $p_i = \frac{f_i}{F}$.
   - Calculate a cumulative probability q_i for each chromosome i (i=1 ,.., pop_size)
     $q_i = \sum_{j=1}^{i} p_j$

   This method requires positive fitness values and a maximization task, so that scaling functions are often used to transform the fitness values accordingly.

2. Stochastic Selection:Goldberg [93]
   - Calculate selection probabilities
   - Successive pairs of individuals are drawn using roulette-wheel selection
   - The string with higher fitness function is selected to produce the next generation.

3. Tournament Selection: Goldberg [93]
   Take a random uniform sample of a certain size $q > 1$ from the population, selecting the best of these q individuals to survive for the next generation, and repeat the process until the new population is filled. This method is computationally efficient and is easy to implement, which is why it is popular.

4. Ranking Selection: Goldberg [93]
   In this method the population is sorted according to objective function value. Chromosomes are then assigned an offspring count that is solely a function of their rank.

3.6 Scaling Methods

If the fitness function is given for each chromosome in a population, there are some extraordinary chromosomes, i.e. chromosomes with a high fitness function, which dominate the selection process during the run of the genetic algorithm and the genetic algorithm will converge to a local optimum. To avoid having such situations Goldberg [93] uses the so called scaling function methods, which try to accentuate differences between individuals in the population and to avoid local optimum.

1. Linear Scaling Goldberg [93] $g = a.f + b$. Where a, b are constant. The role of the coefficients a and b is that they enforce quality of the raw and scaled average fitness values and cause maximum scaled fitness to be a specified multiple of the average fitness. In the linear scaling method the fitness function must be positive otherwise this method does not work well.

2. Logarithmic Scaling $g = a - log(f)$. This function assume a guaranteed positive objective function as well as $a \leq log(f)$

3. Exponential Scaling $g = (a.f + b)^c$. Where a and b are chosen as in the case of the linear scaling methods.

4. Sigma Truncation $g = f - (\overline{f} - c.\sigma)$ Where c is a small integer and σ is the population's standard deviation; In this case the possible negative functions g are set to zero.

3.7 Crossover Operators

Crossover operators play a central role in genetic algorithms. They are used to explore the search space. Crossover operators operates generally in two individuals of the population to create new offspring. These offspring are different from their parents but do inherit some characteristics. The most important role of crossover operators is the disruption. The first crossover operator created by Holland [108] is the one-point crossover operator, which operates on two individuals (bit string). One position is chosen randomly on each individual, then substring are exchanged.

Example :

Parent 1

1	0	0	1	1	0	1	0	1

Parent 2

0	0	1	0	1	1	1	0	0

If we choose the position 5 the offspring will be as follows:

Child 1

1	0	0	1	1	1	1	0	0

Child 2

0	0	1	0	1	0	1	0	1

Several crossover operators were created after the one-point crossover. The new crossover operators depend on the kind of representation used.

3.8 Mutation

Mutation operators were introduced by Holland [108]. They try to change single bits in the new offspring by inverting them.
Example :
$0 \longrightarrow 1$
$1 \longrightarrow 0$
The mutation operator is used with mutation probability P_m, which should be very small.

3.9 Different Kind Of Genetic Algorithms

Since the last three decades several new genetic algorithms have been developed. In this section the most important genetic algorithms are presented as follows:

1. Simple Genetic Algorithm: As described by Goldberg [93], is composed of three main operators:
   -Reproduction:
   In this process individuals are copied according to their fitness function. Individuals with high fitness function have more chance of participating in the production of the next generation whereas individuals with low fitness have less chance. -Crossover:
   This operator takes two parents from the current population and create two children which are different from their parents but contain some characteristics of their parents.
   -Mutation:
   This operator tries a random alteration of children. The goal of this operator is to avoid the quick convergence of the genetic algorithm.

2. Steady State Genetic Algorithm: Syswerda [187] In this algorithm there is one parameter which is the number of new chromosomes to create. In general one or two children are created and inserted in the population.

3. Incremental Crowding Genetic Algorithm: De Jong K. A. [55] and Mahfoud [141]. It follows the simple genetic algorithm except that only a fraction of the population

reproduces and dies with each generation.

4. Parallel Genetic Algorithms:
   In this algorithm multiple processors are used in parallel. Each individual selects by itself. It looks for partner in its neighborhood only [156].
5. Distributed Genetic Algorithms:
   multiple populations are separately evolved with a few interactions between them.
6. Messy Genetic Algorithms: [94]
   It uses a number of "exotic" techniques such as variable-length chromosomes and a two-stage evolution process.
7. Multi-objective Genetic Algorithms: [213]
   If there are many objective functions, the simple genetic algorithm does not work, this is why new genetic called multi-objective genetic algorithm are created.
8. Hybrid Genetic Algorithms:
   This represent genetic algorithms combining with other search algorithms like simulated annealing, tabu search, hill climbing, ..etc to perform their quality.
9. Compact Genetic Algorithm: [101] Where the population is represented as a probability distribution over the set of solutions and is equivalent to the simple GA with uniform crossover.

3.10 Application of Genetic Algorithms

Genetic algorithms are widely applied in areas such as pattern recognition, robotics, artificial life, expert systems, electronic, electrical application, cellular automate, biology, medicine, scheduling and planning problems, as well as others.

Chapter 4

A Review of Representations and Operators for Scheduling Problems

In this chapter an overview of the different kinds of solution representations made for scheduling problems are provided. Firstly, *indirect representations* of a schedule in a chromosome are given, then different *direct representations* of schedules in chromosomes are explained.

4.1 Introduction

Representation is one of the most important characteristics of genetic algorithms, it is in fact the key issue for genetic algorithms. The improvement of a solution by genetic algorithms depends first of all on which kind of representation is used to code the solution, it is very hard to know which representation will give us the best results. In Scheduling problems many representations were proposed by researchers in the last three decades [87], [159], [52], [186], [32], [9], [10], [203], [122], [21], [190], [44], [127],[46],[75], [58], [109], [22], [143]. In this chapter a review of the most used representations in scheduling problems and the corresponding operators are provided. Two kinds of representations are used in most scheduling problems.

- Direct representation, in which the whole schedule is coded in a chromosome, in this representation special and specific operators(crossover and mutation operators) are used.

- Indirect representation, in which just a set of jobs or a set of orders for each job is coded in a chromosome. In this representation we need a decoder, which is also called a schedule builder, to decode a solution (chromosome) into a feasible schedule.

4.2 Indirect Representations of Schedule

In this kind of representation just some elements of schedule such as jobs, machines, processing time, or the plan are represented in a chromosome. To get a schedule from chromosome a schedule builder or decoder is needed.

4.2.1 Binary representation

Nakano [159] was the first person who used binary representation in scheduling problems. In this case the schedule is represented as a binary chromosome, where each chromosome contains 0 or 1. Nakano [159] defined a function called 'prior' which is described as follows: If we consider two jobs j1 and j2 to be scheduled on the same machine then:

$$prior(j1, j2) = \begin{cases} 1 & \text{if the operation } j_1 \text{ is executed before the operation } j_2 \text{ on the same machine} \\ 0 & \text{otherwise} \end{cases}$$

Example: for 3 jobs and 3 machines we have the following tables: 4.1, 4.2.

m1	j2	j1	j3
m2	j3	j1	j2
m3	j2	j1	j3

Table 4.1: Jobs Matrix

Machine sequence:

j1	m1	m2	m3
j2	m1	m3	m2
j3	m3	m1	m3

Table 4.2: Machines Matrix

We define a function prior (j1,j2) on (m1,m2,m3) :

- prior (j1,j2) =0 on m1
- prior(j1,j2)=1 on m2
- prior (j1,j2)=0 on m3

The value of the function prior (j1,j3) on (m1 m2 m3) is :

- prior (j1,j3)=1 on m1
- prior (j1,j3) = 0 on m2
- prior (j1,j3)= 1 on m3

The value of the function prior (j2,j3)on (m1,m2,m3) is :

- prior(j2,j3)= 1 on m1
- prior(j2,j3)= 1 on m3
- prior(j2,j3)= 0 on m2

Through the value of the function prior we determine a 3 x 3 matrix:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

prior(j2,j3)= 0 on m2:

Through the value of the function prior we determine a 3 x 3 matrix:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

prior(j2,j3)= 0 on m2:

Through the value of the function prior we determine a 3 x 3 matrix:

$$\begin{pmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 1 & 1 & 0\end{pmatrix}$$

In this representation a conventional genetic algorithm is used with conventional operators (crossover and mutation) described in [93]. When a conventional crossover operator is used, the chromosomes produced are illegal (they give no feasible schedule). Nakano [159] used an evaluation function to find a legal chromosome g', as similar to the initial chromosome g as possible, then g' is evaluated to determine the fitness of g. In survival of chromosome a treatment is introduced called forcing, it replaces the chromosome g with g' when g is selected as a survivor.

Conventional Operators In [93] the following operators are described as follows:

- One point Crossover

  Example:

  Parent 1:

  $$\begin{array}{|ccccc|ccc|}\hline 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1\\ \hline\end{array}$$

  Parent 2:

  $$\begin{array}{|ccccc|ccc|}\hline 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1\\ \hline\end{array}$$

  Child 1:

  $$\begin{array}{|ccccc|ccc|}\hline 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1\\ \hline\end{array}$$

  Child 2:

  $$\begin{array}{|ccccc|ccc|}\hline 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1\\ \hline\end{array}$$

  This operator works on conventional binary representation. It uses one random cut point on each chromosome, then the chromosomal material is swapped and two different children are created

- Two Point Crossover

  This operator works like the one point crossover [93] , except that two cut points rather than one are selected at random, and chromosomal material is swapped between the two cut point.

Example :

Parent 1:

1	1	0	0	1	1	0	1

Parent 2:

0	1	0	0	1	0	1	1

Child 1:

1	1	0	0	1	0	0	1

Child 2:

0	1	0	0	1	1	1	1

Knowledge-Based Nonuniform Crossover: This Crossover Operator [143] also works on binary representation. The difference between this operator and the conventional crossover operators is that in conventional crossover operator masks, each bit is 1 or 0 with equal probability. In Knowledge-Based Nonuniform Crossover, the crossover mask is a string of real members $\in [0, 1]$. Real member represent a probability of selection of a gene in a chromosome, which depends on the position of the bit in the string and on problem dependent knowledge.

Example:

P1:

1	1	0	0	1	1	0	1

P2:

0	1	0	0	1	0	1	1

Mask:

0.5	0.3	0.7	0.1	0.6	0.2	0.9	0.8

The children are obtained using the following relation:

$Prob(P1(i) = P2(i)) = p_i$

p_i is the probability of the bit i.

4.2.2 Non Binary Representations

In this representation each gene in the chromosome contains natural numbers. Each number corresponds to the job to be scheduled or the processing time of a job on one machine.

Example:

If we consider a set of 9 jobs to be scheduled we represent these jobs in a chromosome as follows:

1	2	3	4	5	6	7	8	9

Non binary representation is a more natural to have a representative schedule than a binary representation. The schedule is considered to be sequencing problem, therefore the same representation and operators used in the *Traveling Salesman Problem (TSP)* are used in scheduling problem. We consider the following representation and operators:

Orders representation [186] The genes of each chromosome are represented as orders to be scheduled. In this case the scheduling problem is considered to be a sequencing problem and we can approach it in the same way we would with the *Travel Salesman Problem.* In the *TSP* the operators do not use any information concerning the distance between cities, this means that it is possible to use these operators for other sequencing problems, like the scheduling problem (the order in the scheduling is very important).

- Partial Matched Crossover [93]

  This operator is used for the ordering representation, a sub-list is mapped onto a sub-list of the other parent and the remaining information is exchanged. This operator works by using the following four steps:

  Step 1 : Choose an interval to be swapped .

  Step 2 : Create a map from the selected interval.

  Step 3 : Exchange the two intervals.

  Step 4 : Use the map to alter the new solutions so that they are once again legal (no conflict) example :

Parent1:

1 2 3	4 5 6 7	8 9

Parent2:

4 5 2	1 8 7 6	9 3

Firstly the segments between the cut (I: the symbol of the cut) are swapped:

O1 =(I 1 8 7 6 I)

O2 =(I 4 5 6 7 I)

We have a set of mapping : $1 \leftrightarrow 4, 8 \leftrightarrow 5, 7 \leftrightarrow 6, 6 \leftrightarrow 7$ The cities for which there is no conflict are filled in:

O1 = (2 3 I 1 8 7 6 I . 9)

O2 = (. 2 I 4 5 6 7 I 9 3)

(The point means until now there is no gene) The rest not filling genes are replaced by the mapping list : (for example : in the first gene of O1, it should be 1 but there is a conflict, it is replaced by 4 using the mapping 1 ” 4). We have the result in the offspring as follow:

O1 = (4 2 3 1 8 7 6 5 9)

O2 = (1 8 2 4 5 6 7 9 3).

- Order Crossover [52]

  A sub-list from one parent is chosen then a relative order of tasks is preserved from another parent. This operator works using two steps :

  Step 1 :Choose an interval to be copied into offspring

  Step 2 :Starting from the last interval of one parent , the tasks from the other parent are copied in the same order , omitting symbols already present.

  Example :

  P1= (1 2 3 I 4 5 6 7 I 8 9)

  P2=(4 5 2 I 1 8 7 6 I 9 3)

  O1=(I 4 5 6 7 I)

  O2=(I 1 8 7 6 I)

  After the removal of the tasks 4 ,5,6,7, which already exist in the first offspring, we

will have:
O1 =(2 1 8 4 5 5 6 9 3)
If we do the same thing for the second offspring, we will have :
O2=(3 4 5 1 8 7 6 9 2).

- Order based crossover [186]
  It randomly selects several positions in a chromosome, then the order of tasks in the selected positions in one parent is imposed upon the corresponding tasks in the other parent.
  Example:
  P1= (1 2 3 4 5 6 7 8 9)
  P2= (4 1 2 8 7 6 9 3 5)
  The positions 3 rd, 4 th , 6 th and 9 th are randomly selected, the ordering of tasks in these positions from P2 will be imposed on parent P1. The tasks are present at position 2 , 5 , 6 , 8.In the offspring the elements from p2 (2-8-6-5) on the positions are recorded to match the order of the same elements from P2 (order: 2-8-6-5) the offspring is a copy of P1 on all positions except positions 2,5,6,8:
  O1 = (1 . 3 4 . . 7 . 9)
  All other elements are filled in the order given in parent P2.
  O1 = (1 2 3 4 8 6 7 5 9)
  By using the same steps the second offspring is:
  O2 = (3 1 2 8 7 4 6 9 5).

- Position based crossover [186]
  Is much the same as the order based crossover, the only difference being is that the position based crossover selects several tasks at random instead of selecting one subsequence of tasks to be copied.

- Edge recombination operator [203]
  In this case the scheduling problem is represented as a sequencing problem. It is considered to be an ordering problem similar to that of the *Traveling Salesman Problem.*

The edge recombination operator does not work on cities but it explores the information on the edges in a tour. The offspring is built from the edges present in both parents (each parent represent a tour). This operator uses an edge map, its stores all the connections from the two parents that lead into and out of a city. After an edge map is made from the two parents, an initial city from one of the two parents tours is chosen (this is the first city in the parents tours). If the two initial cities from the both parents have the same number of edges on the edge map, one initial city is chosen randomly, or else the initial city which has the smallest number of edges on the edge map is chosen. In step 2: all occurrences of the chosen city are moved from the left -hand side of the edge map. Step 3: If the chosen city has entries in its edge list, go to step 4, or else go to step 5. Step 4: Determine which of the cities in the edge list of the current city has the fewest entries in its own edge-list. The city with the fewest entries is consider as a chosen city, go to step 2. Step 5: If there are no remaining unvisited cities then stop, or else randomly choose an unvisited city and then go to step 2.

Example:

P1 = (1 2 3 4 5 6)

P2 = (2 4 3 1 5 6)

first we make the edge map:

1: 2 6 3 5

2: 1 3 6 4

3: 2 4 1

4: 3 5 2

5: 4 6 1

6: 1 5 2

The initial cities of the parents are: 1 and, 2, and we have to choose one of them. City 1, and 2 have the same number of edges, so we can choose one city randomly. City 2 is chosen:

1: 6 3 5

* 2: 1 3 6 4

3 : 4 1

4: 3 5

5: 4 6 1

6: 1 5

The edge list indicates there are four cities to be candidate for the next chosen city (1,3,6,4). City number 2 is eliminated, and city 3 is chosen randomly. City 3 has edges to cities 4 and 1 . Next city 4 is chosen (It has the fewest edges). City 4 has an edge to 5, and then city 5 is chosen. This city has edges to 1 and 6, we chose city 1 at random because it has the same number of edges as city 6. City 1 give way to city 6, and the offspring is as follows:

O= (2 3 4 5 1 6).

This offspring is composed entirely of edges taken from the two parents (P1 and P2).

- Precedence preservative crossover which was applied to scheduling problems by [22]. The offspring chromosome is initially empty. Then a vector of length 'n' is randomly filled with elements of the set 1,2. This vector defines the order in which genes are drawn from parent 1 and parent 2 respectively .After a gene is drawn from one parent and deleted in the other one , it is appended to the offspring chromosome.
  Example:

  P1:
  3 2 2 2 3 1 1 1 3

  P2:
  1 1 3 2 2 1 2 3 3

  gene of parent:
  1 1 2 2 2 2 1 1 1

  PPX:
  3 2 1 1 2 1 2 3 3

- Maximal Preservative Crossover

  This operator was introduced by [155]. A substring is randomly selected from the first parent whose length is greater than or equal to 10 and smaller or equal than the size divided by 2. (This choice is done to preserve more information from the parents). Then all the chosen elements from the first parent are removed from the second parent. After that, the substring chosen from the first parent is copied onto the first part of the offspring. Finally, the rest of the offspring is filled up with elements in the same order as they exist in the second parent.

  Example

  Parent 1

1	2	3	4	5	6	7	8

  Parent 2

2	4	6	8	7	5	3	1

  The substring (3 4 5) is chosen from the first parent.

  The offspring will be:

3	4	5	2	6	8	7	1

- Uniform Order-Based Crossover This operator is described in [43] as follows:

  This operator used the uniform crossover mask to select jobs from parents. If the mask is 1 then the job from parent 1 is selected to be in the offspring and the remaining jobs are filled in the offspring in the same order they appear in Parent 2.

  Example:

  Parent 1

1	2	3	4	5	6	7	8

  Parent 2

2	4	6	8	7	5	3	1

Mask:

0	1	1	0	1	0	1	0

Offspring:

4	2	3	6	5	8	7	1

4.2.3 Matrix representation

In this representation [87], the scheduling problem is considered to be a sequence problem (the same as a TSP). If we have a sequence of jobs, this sequence is represented by boolean (0,1) matrices that represent predecessor and successor relationships.

Example: If we have the sequence [2 1 3 5 4], then the job 1 precedes job 3, job 3 precedes job 5, and job 5 precedes job 4, etc. This sequence can be represented by the following matrix as follows:

	1	2	3	4	5
1	0	0	1	1	1
2	1	0	1	1	1
3	0	0	0	1	1
4	0	0	0	0	0
5	0	0	0	1	0

Table 4.3: [2 1 3 5 4]

The matrix element $m_{ij} = 1$ if and only if job i occurs before job j in the sequence. The sum in the column determines the number of predecessors of a job, and the sum in the row determines the number of successors of a job . The matrix representation has three properties:

1. The number of 1 is exactly: $n(n-1)/2$

2. mij=0 for all 1 <= i <= n (no cycle exists)

3. if mij =1 and mjk = 1 then mik = 1
   (the transitive nature of the order must be respected).

Operators for Matrix Representation

- Union Operator [87] The union operator works in 4 steps. If we have a binary matrix representing the parent sequences, first partition the set of symbols (jobs) into two disjointed sets. Second, construct the matrix which contains the bits from the first parent that defines the relationships within the first subset of jobs and construct the matrix which contains the bits from the second parent that defines the relationships within the second subset of jobs. Third, perform the logical OR (union) of these two matrices resulting in a matrix that contains unique attributes from both parents but with some attributes still undefined. Fourth, convert the obtained matrix into a sequence of jobs, which corresponds to the created children.

  Example:
  If we have the two sequences:
  Parent 1: [2 1 3 5 4]
  Parent 2: [5 1 2 3 4]
  and if we choose the disjointed subsets: [2 1] and [3 5 4].
  Then the two parents can be represented by the binary matrix as follows:
  Parent 1: [2 1 3 5 4]

	1	2	3	4	5
1	0	0	1	1	1
2	1	0	1	1	1
3	0	0	0	1	1
4	0	0	0	0	0
5	0	0	0	1	0

  Parent 2:[5 1 2 3 4]

	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	1	0
3	0	0	0	1	0
4	0	0	0	0	0
5	1	1	1	1	0

The Matrix resulting from the union of the two matrices above is as follows:
Children:

	1	2	3	4	5
1	0	1	X	X	X
2	1	0	X	X	X
3	X	X	0	1	1
4	X	X	0	0	0
5	X	X	1	1	0

- Intersection Operator [87] The idea behind the intersection operator is that the characteristics that are common to two good solutions should be passed onto the children. Given two matrices from the parents, a logical AND (intersection) of the two parents is used to create new children.

  Example:

  If we have the two sequences:

  Parent 1: [2 1 3 5 4]

  Parent 2: [5 1 2 3 4]

  The two parents can be represented by the binary matrix as follows:

  Parent 1:[2 1 3 5 4]

	1	2	3	4	5
1	0	0	1	1	1
2	1	0	1	1	1
3	0	0	0	1	1
4	0	0	0	0	0
5	0	0	0	1	0

Parent 2:[5 1 2 3 4]

	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	1	0
3	0	0	0	1	0
4	0	0	0	0	0
5	1	1	1	1	0

The Matrix resulting from the intersection of the two matrices above is as follows:

Children:

	1	2	3	4	5
1	0	0	1	1	0
2	0	0	1	1	0
3	0	0	0	1	0
4	0	0	0	0	0
5	0	0	0	1	0

The children corresponding to the matrix can formulate the sequence:

Children: [1 2 3 5 4]

Compared to other operators like Partial Matched Crossover or Edge Recombination operator, the genetic algorithm with Union or Intersection operators computes slowly [87].

4.2.4 Permutation representation

n jobs (tasks) are executed on a set of m machines [22]. We have to assign all jobs to machines and sequence the assigned jobs for each machine such that an overall cost-function is minimized. p1,.,pn1, pn1+1,.,pn2,..,pnm-1 +1,..,pn

machine 1 machine 2 machine m

This representation is the permutation of the job numbers 1,2,..,n, which is m-partitioned by the numbers nk.

Crossover Operators In this representation two kinds of crossover operators are used, namely the General *Order Crossover (GOX)* and the General *Partial Matched Crossover(GMPX)* [22].

Example:

Parent1:

3	2	2	2	3	1	1	1	3

Parent2:

1	1	3	2	2	1	2	3	3

GOX offspring:

1	3	2	2	2	3	1	1	3

GPMX offspring:

1	3	2	2	3	1	2	1	3

4.2.5 Order/process plan representation

In [9] and [10], each gene of a chromosome contains an allele representing an order/process plan.

Example:

oder1/planA	order3/planB	..	order2/planA	order5/planC

4.2.6 Order /process plan /resources representation

In [9] and [10], each gene of a chromosome is represented as follows:

Op0 Op1	Op0 Op1 Op2	
Job: 1	Job: 0	..
M2 M1	M0 M1 M0	

The order of the jobs is important, this means that the priority of job 1 is bigger than the one of job 0.

The operators used in this case are :crossover operators, plan crossover, mutation operators.

4.2.7 Time-dependent preference lists representation

To encode the scheduling problem, there is for each machine (resource) a list of preferences which are linked to times [52]. For example : Resource 1: (40 o3 o1 o2 'wait' 'idle'),the first element of a list mean the time at which the orders work. The machine (resource) should execute the order o3 before o1. If this is not possible, then it should execute the orders o1 and o2. If no order is available to execute the machine should wait. Three operators are used in this case: run-idle: It inserts the 'idle' as the second member of the preference list and resets the first member (time) of the preference list to 60 minutes . scramble: it scrambled the members of preference list . crossover: The operator exchanges preference lists for selected resources.

4.2.8 Preference list-based representation

This representation was originally created by [52]. If we have m machines, n jobs, each chromosome contains m sub chromosomes, and each sub chromosome contains n operations, each operation has to be processed on the relevant machine. The sub chromosomes are, in reality, a list of preference, this means each machine has its own preference list. Example: with 3 machines(m1,m2,m3) and three jobs(j1, j2, j3), we consider a chromosome:

ch = [(2 3 1)(1 3 2)(2 1 3)] The gene (2 3 1) is the preference list for the machine m1 ,the gene (1 3 2) is the preference of the machine m2, and the gene (2 1 3) for the machine m3. This means the first operations which are preferred are j2 on machine m1, j1 on machine m2 and j3 on machine m3.

4.2.9 Priority rule-based representation

A chromosome is represented as a sequence of dispatching rules for job assignment [75]. If we have n jobs which are executed on m machines, the chromosome is encoded as a string of n x m entry $(P_1, P_2, , P_{nm})$.Each entry Pi represents one rule.

Example : The chromosome ch is represented as follows:

ch = [1 2 2 1 4 4 2 1 3] , each gene is considered as rule. 1:means select an operation with the shortest processing time.
2:means select an operation with the longest processing time .
3:means select an operation for the job with the most total processing time remaining . 4: means select an operation for the job with the least total processing time remaining.

4.2.10 Disjunctive graph-based representation

the job shop is considered as a disjunctive graph $G = (N, A, E)$ defined as follows: N contains nodes representing all operations, A contains arcs connecting consecutive operations of the same job, E contains disjunctive arcs connecting operations to be executed by the same machine . The chromosome is represented as a binary string, which corresponds to an order list of disjunctive arcs in E. X_{ij} = {1, settle the orientation of the disjunction arc from node j to node i 0, settle the orientation of the disjunction arc from node I to node j } This representation gives an infeasible schedule because of the possibility of having a cyclic graph.

4.2.11 Machine based representation

Each gene of a chromosome is represented as a machine, ch = (m1, m2 ,..,mm)

4.2.12 Operation-based representation

[44], Each gene is represented as an operation O_{jim}, which means the i-th operation of job j is executed on machine m. The chromosome is represented as follow: CH = [O211 O111 O122 O133 O223 O 232 O312 O321 O333] m x j genes are in the chromosome.

4.2.13 Job-based representation

Each gene of a chromosome is represented as a job [44], [22]. In the sequence of jobs existing in the chromosome, all operations of the first job are scheduled first and then the second one is scheduled and so on. The occurrence of the operations in job j is very important (the first operation in job has the best available processing time for machine m). New Crossover was introduced, this operator is called precedence preservative crossover [22].

4.2.14 Completion time based representation

Each gene of a chromosome represents a completion time of operation , for example :

ch = [c111, c122, c133, c211,c223,c232,c312,c321,c333]

where Cjir means the completion time for the operation i of job j on machine r. This representation gives an illegal schedule , this why a special crossover and operation is used for this representation.

4.2.15 Random key representation

Each gene of a chromosome is encoded as a random number [16]. For m machines and n jobs, each gene contains two parts: an integer between 1 and m and a fraction generated randomly between [0,1]. The integer part represent the machine assignment for the job.
Example:
ch = (1 .34 1 .09 1 .88 2 .66 2 .91 2 .01 3 .23 3 .213.44).

4.2.16 Pigeon-Hole Coding

The idea of this representation [122] is to convert the permutation in a sequencing problem into another form in order to facilitate the application of crossover operators. For example, consider a permutation p=[1 2 3 4]. The four elements are considered to be a flock of four numbered pigeons lining up to move into a new home one after another with the four pigeon holes arranged in order and labeled as [1,2,3,4]. If the pigeons are free to make their own choices by selecting one hole from the empty holes available at that time, then the order of these pigeons in the pigeon holes could be used to represent a particular permutation. For example, the first pigeon chooses the second hole out of the four empty pigeon holes (i.e.

hole 2), the second pigeon selects the second hole out of the remaining three holes (i.e. hole 3), the third pigeon picks the first hole of the remaining two (i.e. hole 1), and the final pigeon takes the last one (i.e. hole 4) . The relative order of the holes being occupied by the pigeons (i.e. [2,2,1]) can be used to represent a permutation of [2-3-1-4] . The mapping between the permutation and the representation is one to one correspondence, And the permutation of [2-3-1-4] can be coded by a chromosome of three integers [2,2,1]. We can generally formulate this kind of encoding : To encode a solution of a sequencing problem $[p1, p2, .., pn]$ where $pi, i = [1, n] andpi = pjifi = j$, the pigeon-hole coding schema requires a chromosome of n-1 integer genes $[p1, p2, .., pn - 1]$, where $pl = [1, n - i + 1]$. The value of the k th gene is given by the following rule:

$$\begin{aligned} P_1 &= \pi_1 && (4.2.1) \\ P_k &= \pi_k - \sum \varphi_k(\pi_i) && (4.2.2) \\ Where & \\ \varphi_k(\pi_i) &= \begin{cases} 1 & \text{if } \pi_i < \pi_k \text{ for } k > 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

This kind of coding schema has the following properties: n-1 integer genes are required for a sequencing problem of n elements. permitted values of the i-th gene are [1, n-i+1]. .offspring produced from mutation are valid, provided the value of the varied gene is within permitted values .

4.2.17 New Genetic Representation

A new representation was used in [190], where each chromosome represent one resource and one individual corresponds to a set of chromosomes. Each chromosome is decodes to the sequence of jobs to be scheduled on one resource. The number of chromosomes in one individual equals to the number of resources in a schedule.

1	2	3	4	5	6	7	8

Common Cluster Crossover In this crossover operator [190] the crossover points are not chosen randomly, but this operator identifies the clusters(genes linked together) of sub-sequences and exchanges the common clusters between two parents.

Example:

Parent 1:

Chromosome:

1	7	5	4	2	2	1	0	1	0

Sequence:

2	9	8	3	7	4	6	5	10	1	11

Parent 2:

Chromosome:

1	0	5	2	1	0	1	0	2	1

Sequence:

2	1	8	6	4	5	3	7	11	9	10

Offspring:

Chromosome:

1	7	5	2	1	0	1	0	1	0

Sequence:

2	9	8	6	4	5	3	7	10	1	11

4.3 Direct representation of schedule

The Data Structure is used as schedule itself [32]. No decoding or schedule builder is needed. Bruns [32] was the first researcher who employs Genetic Algorithms with direct representation to solve production scheduling problems. This case was used for one operation orders with associated machine and a start time.

Order x	order y
m1	m1
start x	start y

This representation is available just for one -machine problems. It is not possible to generalize it to n-operation orders or multi-machines.

4.3.1 Knowledge augmented genetic algorithm for scheduling

This direct representation was developed for more complicated scheduling problems (m-machines, n-operation orders ..). The schema of this representation is as follow:

op 7 B1	op7 B2	op 4 A1	op4 A2
m9	m3	m6	m1
[10,15]	[16,17]	[2,9]	[11,19]

Advanced Crossover As described in [32] this operator tries to select the non delayed orders from one parent which are inherited from offspring and the missing orders are selected from the second parent (first, non delayed orders are selected according to their increasing start time, then the delayed ones are selected).

Advanced Mutation [32] One order is selected randomly, then the process plan of the corresponding order is changed. After that an operation is selected randomly and an alternative machine for the operation is chosen. Finally an operation is selected randomly and the corresponding time is changed to the earliest possible time.

Chapter 5

Design and Integration of Genetic Algorithms in the existing DéjàVu Class Library

In this chapter the integration of genetic algorithms in the *DéjàVu class framework*, which is an application framework [80] and [63] , is described. The Unified Modeling Language(UML) is used to help us in software analysis and design. First, an overview of the Unified Modeling Language (UML) is given then a description of designing the integration of genetic algorithms in *DéjàVu Class framework* is given using class diagrams and sequence diagrams.

5.1 Introduction

The purpose of the Conception and the integration of genetic algorithms in the existing *DéjàVu* scheduling framework [63] is to find near optimal solution or the best solution for the scheduling problem. To design an efficient genetic algorithm for the scheduling problem we should first design the representation of the solution from an initial schedule, this means finding which representation gives us the best solution for the scheduling problem.

These representations are called chromosomes or genomes in genetic algorithm terminology. Second, genetic operators are chosen to create new offspring during the evolution of the genetic algorithm. In this thesis, several genetic algorithms are tested and compared to each other to see which operator gives us a better solution. The problem is how we integrate genetic algorithms in the existing DéjàVu system? To answer this question, we should analyze and design how genetic algorithms communicate with other components of the system. To do that, we work with the Unified Modeling Language (UML), which helps us to illustrate the analysis and design of a system. The UML [166], [129], [25] is a language used to specify, visualize, and document the artifacts of an object-oriented system under development. It is an attempt to standardize the artifacts of analysis and design: semantics models, syntactic notation, and diagrams.

We used as a design tool the Rational Rose [166]. DéjàVu scheduling is a semi-finished application framework [80], which was designed by Dorn et al. [67], [66], [63].

5.2 Overview Of Unified Modeling Language

The Unified Modeling Language (UML) [166], [129] is a modern approach that helps software designer to analysis and design complex software. UML as described in [25] is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. UML is not a programming language but it is a modeling language. From UML we can generate a code in different programming languages like C++ or Java this is called forward engineering [25] . We can also generate an UML model from a programming language, this is called Reverse engineering [25].

The conceptual model of the UML is composed of 3 major elements:

- The Basic Building blocks of UML

  UML is composed of 3 kinds of building blocks:

  1. Things: There are four kind of things in the UML

     – Structural things

       These represent the static parts of UML, these things represent elements that are either conceptual or physical. In UML we have seven basic things: class, interface, collaboration, use case, active class, component, node.

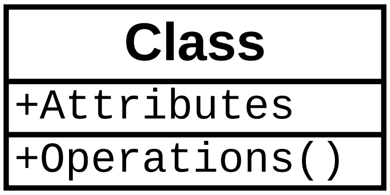


Figure 5.1: Class.

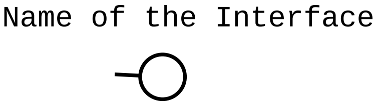


Figure 5.2: Interface.

A **class**, figure 5.1 is a description of a set of objects, which share the same attributes, operations, relationships, and semantics:

An **interface**, figure 5.2 is a collection of operations, which specify a service of a class.

A **use case**, (figure 5.3) is a description of a set of sequences of actions that a system performs that yields an observable result of value to a particular actor. An actor can be a user or another system.

– Behavioral things

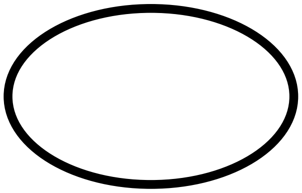


Figure 5.3: Use Case.

Figure 5.4: Messages.

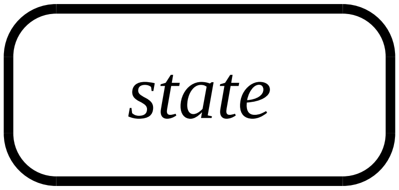


Figure 5.5: State machines.

These are the dynamic parts of UML. These things represent behavior over time and space. There are two kinds of behavioral things.
First, an **interaction**, (figure 5.4), which represents a set of messages exchanged among a set of objects.

Second, a **state machine**, (figure 5.5), which specifies a sequence of states that an object goes through during its lifetime in response to events.

– Grouping things
  A **package**, (figure 5.6) is general-purpose mechanism for organizing elements into groups. Structural things, behavioral things, and other grouping things can be placed in a package.

– Annotation things
  Notes, (figure 5.7) are the explanatory parts of UML models. Notes are comments that are applied to describe or illuminate the elements of the model.

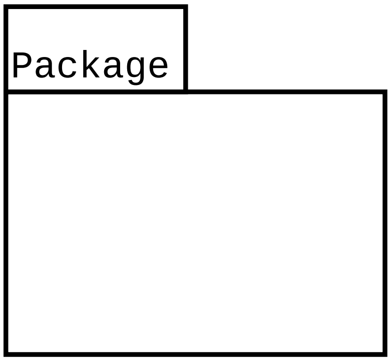


Figure 5.6: Packages.

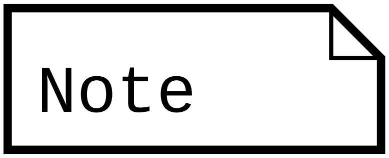


Figure 5.7: Notes.

2. Relationships

   Dependency, (figure 5.8) is a using relationship that states that a change in specification of one thing(like a class Event) may affect another thing that uses it(like class Window).

   Association, (figure 5.9) is a structural relationship that exists in *class diagram* and specifies that objects of one thing are connected to objects of another. An association that is usually made between two classes.

   Generalization, (figure 5.10) is used in a *class diagram* and it corresponds to the relationship between a general thing (called the superclass) and a more specific kind of that thing (called subclass or child).

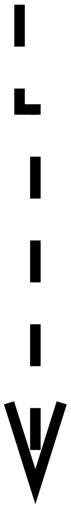


Figure 5.8: Dependency.

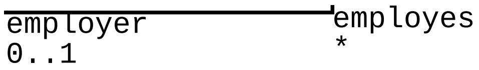


Figure 5.9: Association.

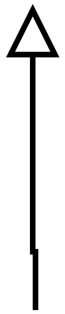


Figure 5.10: Generalization.

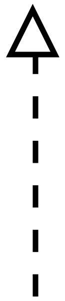


Figure 5.11: Realization.

Realization, figure 5.11 is a semantic relationship between *classifiers*, where one classifier specifies a contract that another *classifier* guaranties to carry out.

3. Diagrams

- The rules, which tell us how the building blocks are connected.
- Some common mechanisms that apply throughout the UML.

Several Diagrams used in UML are:
Class Diagrams, Object diagrams, Use case diagrams, Sequence diagrams, Statechart diagrams, Activity diagrams, Component diagrams, and Deployment diagrams. We will only be explaining the type of diagrams which are used in our analysis and design. If the reader is interested in details concerning the rest of the diagrams see [25].

5.2.1 Class Diagrams

Class diagrams represent the static design view of a system. They shows a set of classes (see figure 7.5), interfaces, and collaborations and their relationships. Class diagrams are the most used in modeling object oriented systems.

5.2.2 Sequence Diagrams

Sequence Diagrams represent the dynamic design view of a system. They show an interaction, consisting of a set of objects and their relationships, including the messages that may be dispatched among them. Sequence diagrams are intersection diagrams, which emphasize the time-ordering of messages.

5.2.3 Use Case Diagrams

A use case diagram shows a set of use case and actors (a special kind of class) and their relationships. Use case diagrams show the static use case view of a system. They are important in organizing and modeling the behaviors of a system and the interaction of a system with its environment.

5.3 Integration Of Genetic Algorithms in DéjàVu Framework

In this section we will make an analysis and design of the integration of genetic algorithms in the existing DéjàVu framework [67], [66], [63] using Unified Modeling Language (UML).

5.3.1 The existing DéjàVu framework

The *DéjàVu framework* contains theoretical classes like schedule class, order class, job class, operation class, resource class, and allocation class, all of which are abstract classes and represent the theoretical part of the framework. The design of the existing DéjàVu framework was directed by the following criteria [63]:

- The scheduler's evaluation of a schedule is based on the evaluation of individual constraints and their weighted aggregation.
- The user has full control over the scheduling process with the ability to experiment with different settings
- The scheduler applies iterative improvement methods to optimize solutions.
- The framework should be extensible and refinable.

The DéjàVu framework is a reusable framework based on the principle of object-oriented design with the support of some well-known design patterns [89], such as:

- The abstract factory
  A domain dependent schedule object is created without specifying its concrete class.
- The factory method
  A domain-dependent order is created in the domain-independent order director class.
- Chain of responsibility
  This gives the most specific or selected user interface element the chance to react on a user action and to pass it onto its supervisor if the element does not know how to react.
- Command
  A user action is encapsulated as an object that has a common interface for undoing or redoing the action.
- Iterator
  Provides a way to access elements of an aggregate object such as a list of constraints sequentially without knowing its implementation.
- Observer
  When an object, such as a schedule changes, all dependents can be notified without having to call all the dependents by using the scheduling object.

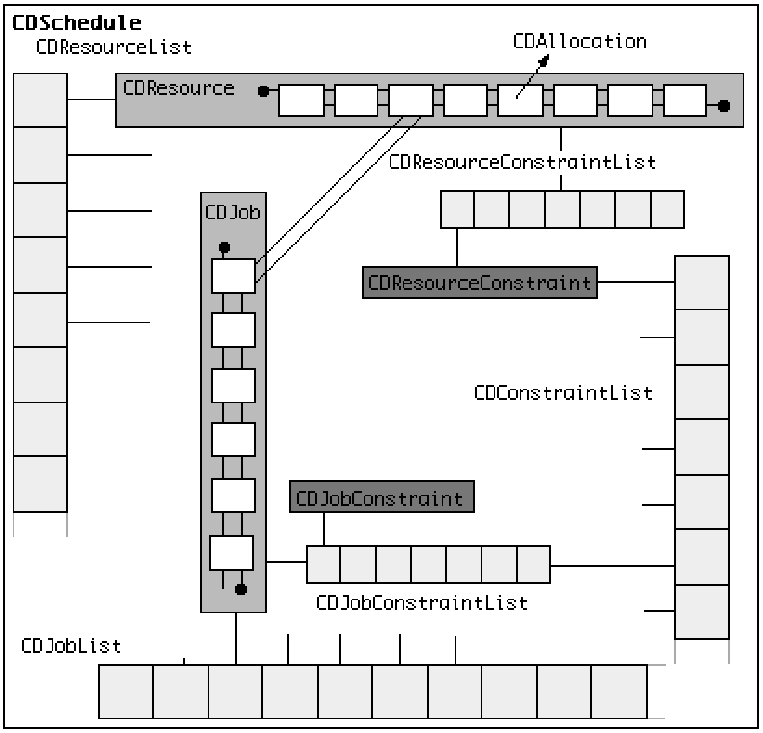


Figure 5.12: Structur of a Schedule

To support a reuse in the *DéjàVu framework* the following principles are used:

1. Scheduling Core. The principle object in the scheduling core is a schedule see figure5.12, which consists of the following concepts:
   - a list of resources with schedule allocations,
   - a list of jobs with their operations,
   - a list of constraints.
2. Schedule Evaluation

   In DéjàVu framework the evaluation of a schedule is defined by the evaluation of individual constraints. The constraints are stored in a constraint list. There are four base constraints in existence; *allocation constraint*, *job constraint*, *resource constraint*, and

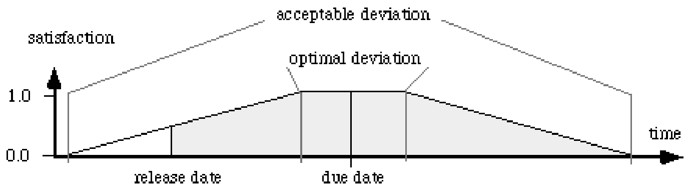


Figure 5.13: Degree of Satisfaction

schedule constraint. Each of these constraints is considered to be abstract class in the *DéjàVu framework.* If a new constraint is created it will be considered to be a concrete class and it is derived from one of the four abstract classes. To evaluate a schedule, each concrete constraint has a satisfaction degree, which is a value between 0 and 1. This satisfaction degree [70] tells us how this constraint influences the evaluation of a schedule. There exist several concrete constraints like *tardiness constraint*, *makespan constraint*, and *compatibility constraint* with two specializations *chemical constraint* and *format constraint.* The *compatibility constraint* tell us how optimal the schedule is if we make the operations in a certain order.In the *compatibility constraint* it is important to find sequences that incorporate infiltration with small percentage.

3. Graphical User Interface

   In DéjàVu framework the Graphical User Interface (GUI) plays an important role. The goal of the GUI is to give the user control over the schedule activities, this means when the system gives us a schedule with poor results, the user has the possibility of changing the schedule to a better one.In the following, the most important Graphical User Interfaces which exist in the DéjàVu framework are described:

   - The supervision hierarchy,
   - Scheduling Tasks,
   - Graphical Schedule Visualization,

4. Algorithms, which try to find better schedule

   In *DéjàVu framework* there are algorithms which optimize the schedule, these algorithms are designed as classes. Until now the only algorithms in existence were: *the* *tabu search algorithm, the simulated annealing algorithm, and the iterative deepening.* The new algorithms are *genetic algorithms*, which are the scope of this thesis. These algorithms are easily combined by deriving them.

5.3.2 Integration of Genetic Algorithms in the DéjàVu framework

To show how the integration of genetic algorithms in the *DéjàVu* framework are designed, a UML (Unified Modeling Language) representation [166], [129] is used to explain our design. Two kind of diagrams are presented, the class diagrams which represent the statical view of our design and the sequence diagrams which represent the dynamic view of our design. To allow the software reuse principle, an existing class library *GAlib library* of genetic algorithms components, developed by Wall [145], [146] is used which contains the basic classes, that are needed to build genetic algorithms (see figure 5.14). The *GAlib library* was used because of the simplicity of reusing its classes by deriving new ones.

Each kind of genetic algorithm represents a class which is a sub class of the class **CDScheduledirector** from the *DéjàVu framework* and the class Genetic algorithm from *GAlib library* [145], [145]. In this thesis two kinds of genetic algorithms classes are integrated, the simple genetic algorithm class [93] and the steady state genetic algorithm class [187]. We can integrate other classes representing new genetic algorithms using the principle of generalization. In figure 5.15 a component diagram is presented showing how genetic algorithms component is connected to other components from *DéjàVu framework* and from *GAlib library.*

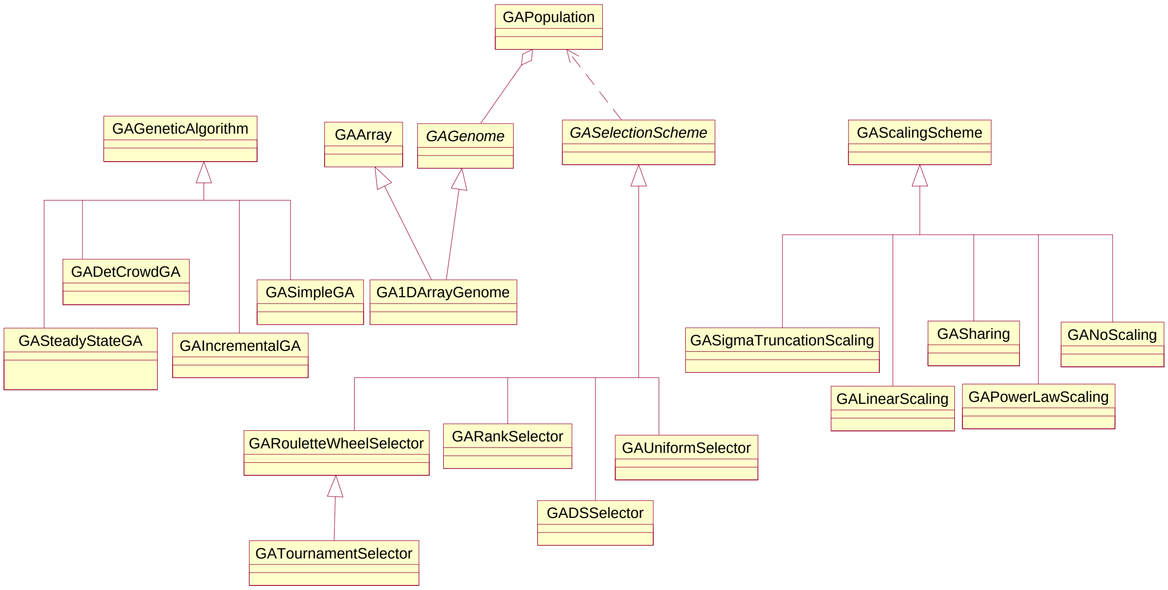


Figure 5.14: The Class Hierarchy of the GALib

Class Diagram of Genetic Algorithm

In figure 5.16 a class diagram shows how the genetic algorithms are integrated in the *DéjàVu framework*. Two principle classes are integrated, the so called **SimpleGA** class, (see figure 5.17) and the **Genome** class, (see figure 5.24).

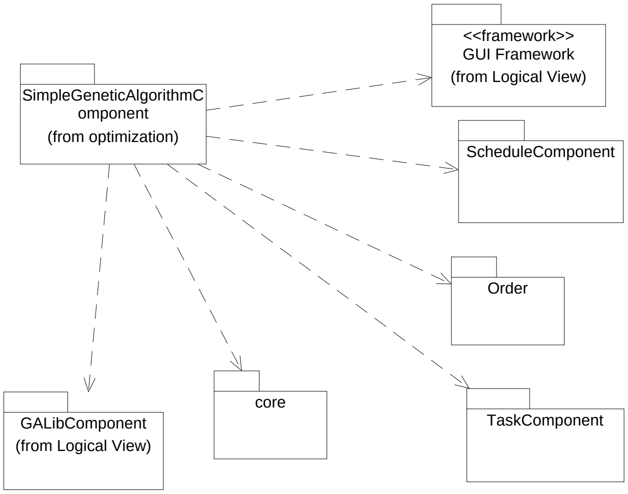


Figure 5.15: Component Diagram: Genetic Algorithm

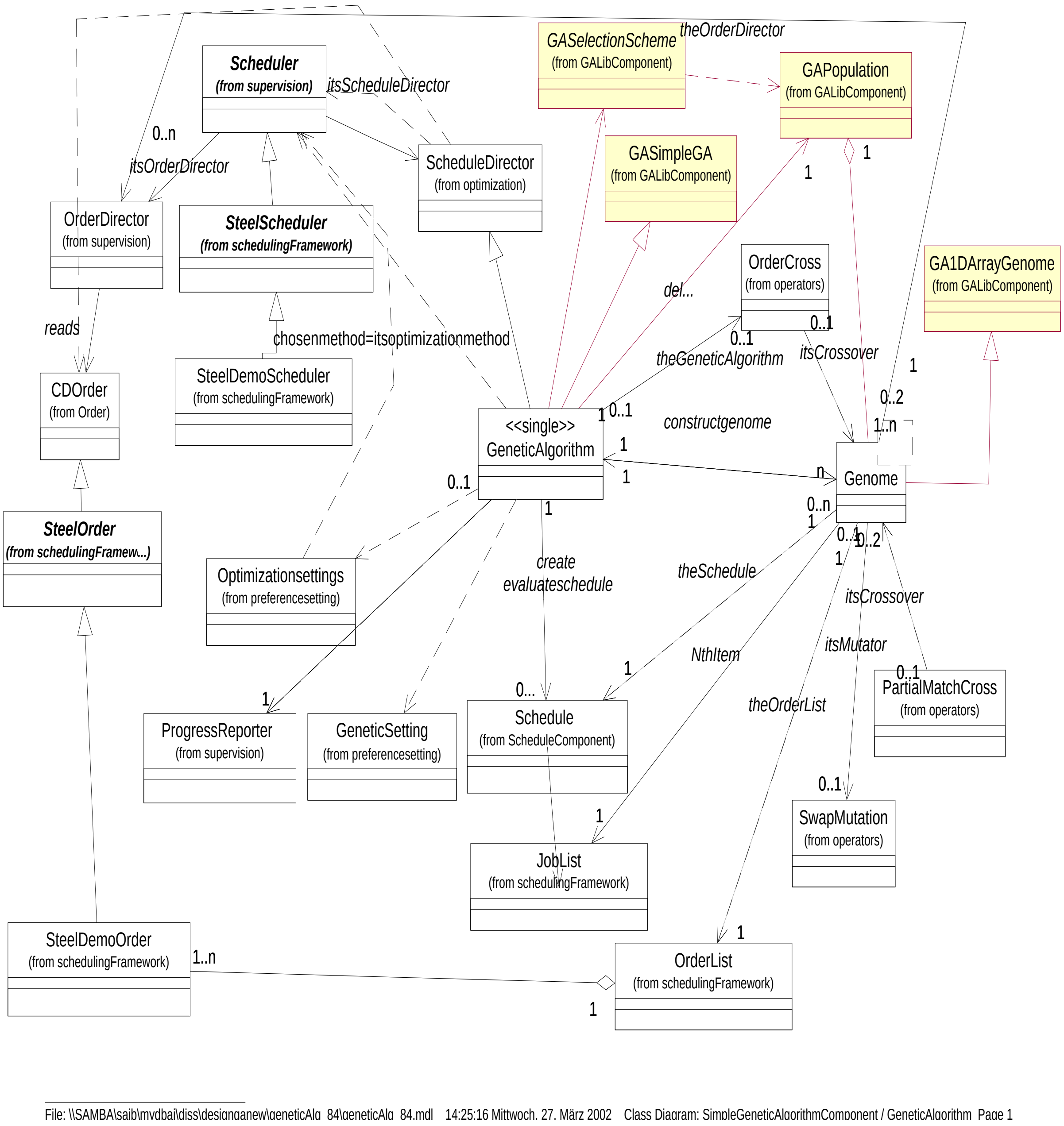


Figure 5.16: Simple Genetic Algorithm

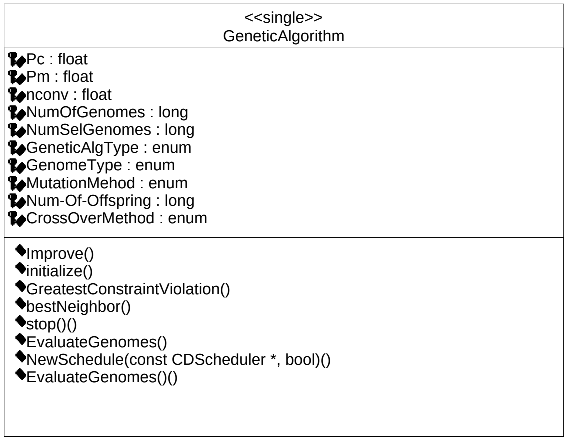


Figure 5.17: Genetic Algorithm Class and its Interfaces

The **SimpleGA** class, (figure 5.17) contains the follwing methods:

- *Initialize()*:

  This method permits us to initialize the genetic algorithms, (see figure 5.18). There are two possibilities, first we can randomly create an initial population, and second we can randomly create a mixture of individuals with one individual from an intial schedule, who was created by heuristic methods from the *DéjàVu* framework.

- *Bestneighbor()*:

  This method is the scope of genetic algorithms, (see figure 5.19). It permits us to create one generation of the genetic algorithm.

- *Improve()*:

  This method is a repeat of the BestNeighbor() method until the stopping criteria are satisfied, (see fig. 5.21).

- *Stop()*:

  This method is responsible for stopping the genetic algorithm.

- *EvaluateGenomes()*:

  This method evaluates each new created genome, (see figure 5.20). The evaluation is done in two parts. First, the genome is decoded to a schedule which is evaluated by an evaluation function, existing in the *DéjàVu* framework, and second, the evaluation function is assigned to the corresponding genome.

- *NewSchedule(const CDScheduler *, bool)*

  This method allows us to create a new schedule

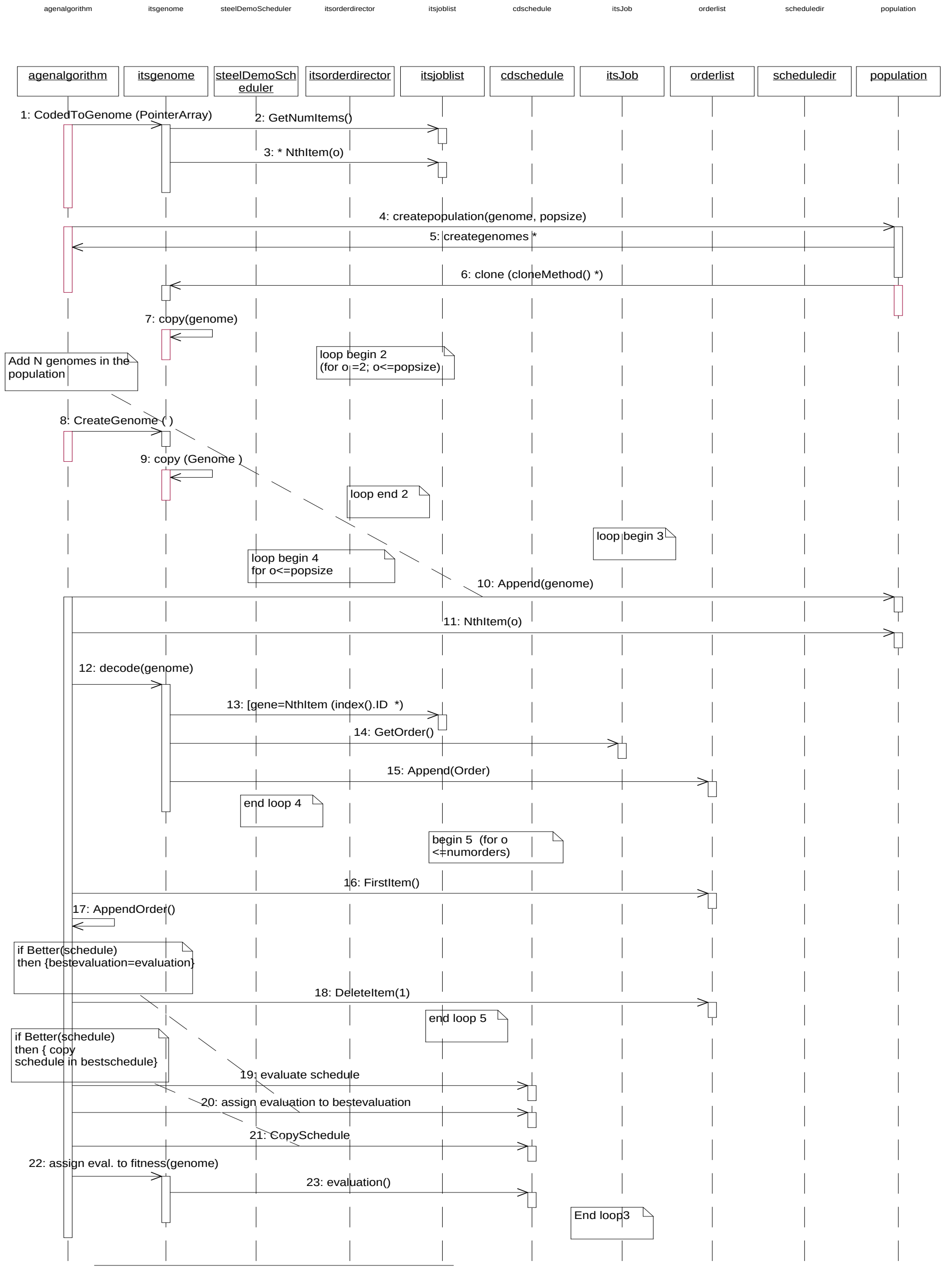


Figure 5.18: Sequence Diagram of the method Initialize in genenetic algorithm class

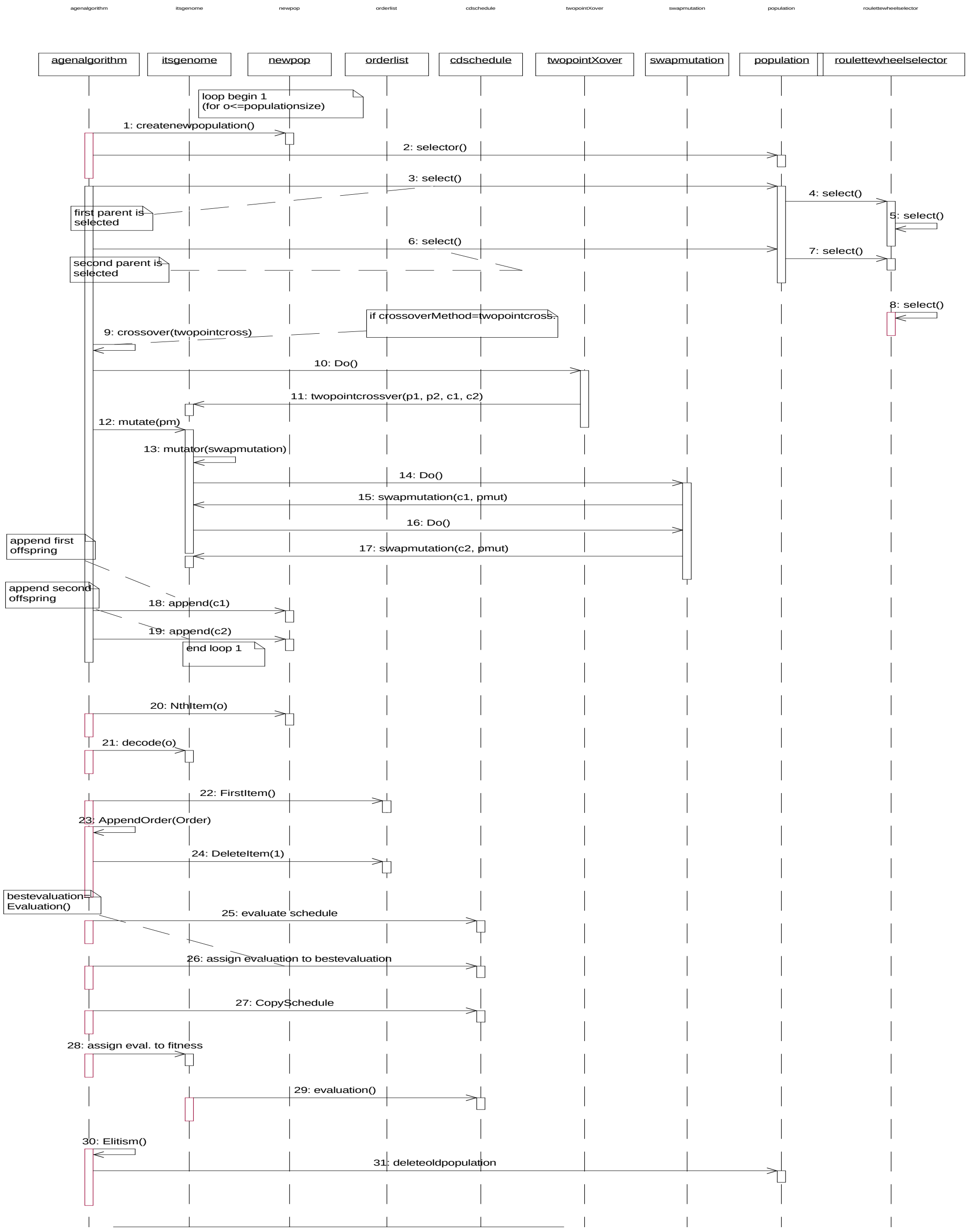


Figure 5.19: Sequence Diagram of the method Best Neighbor in genenetic algorithm class

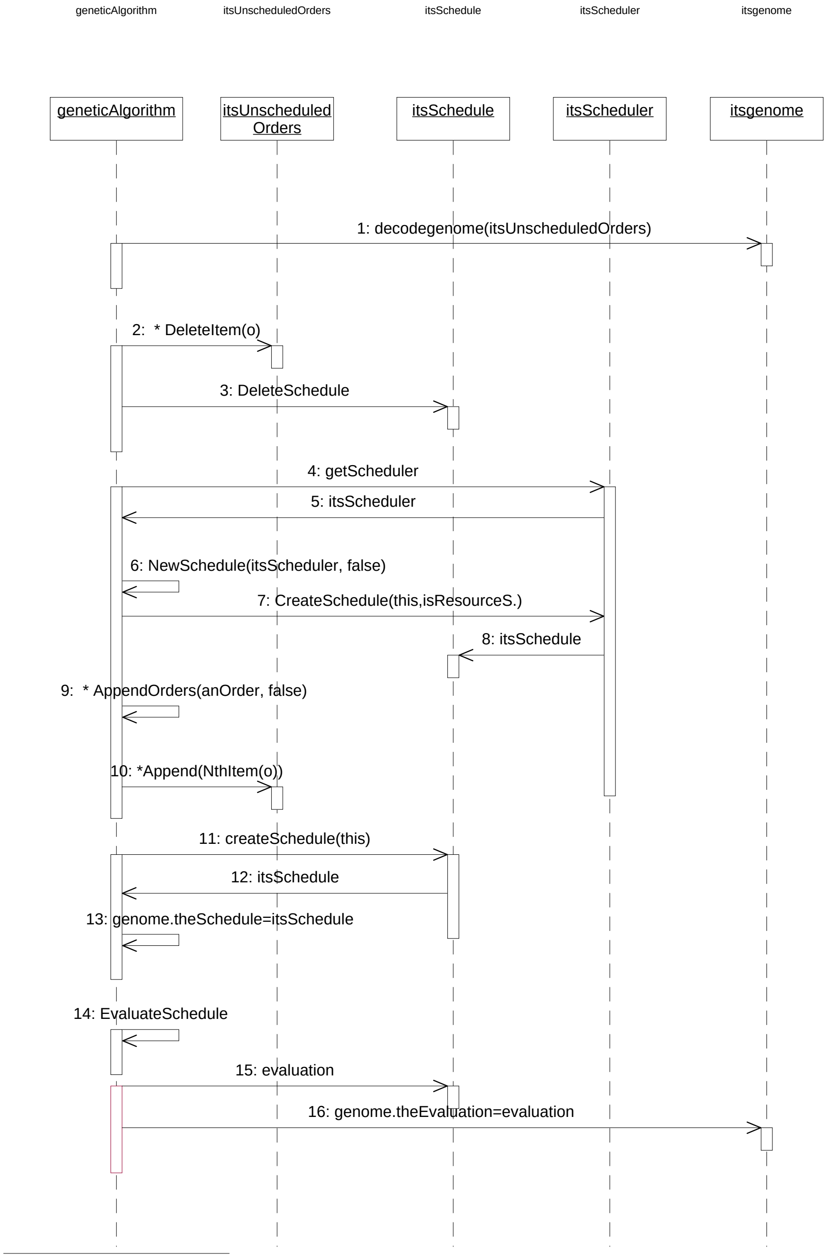


Figure 5.20: Sequence Diagram of the Evaluate genomes method

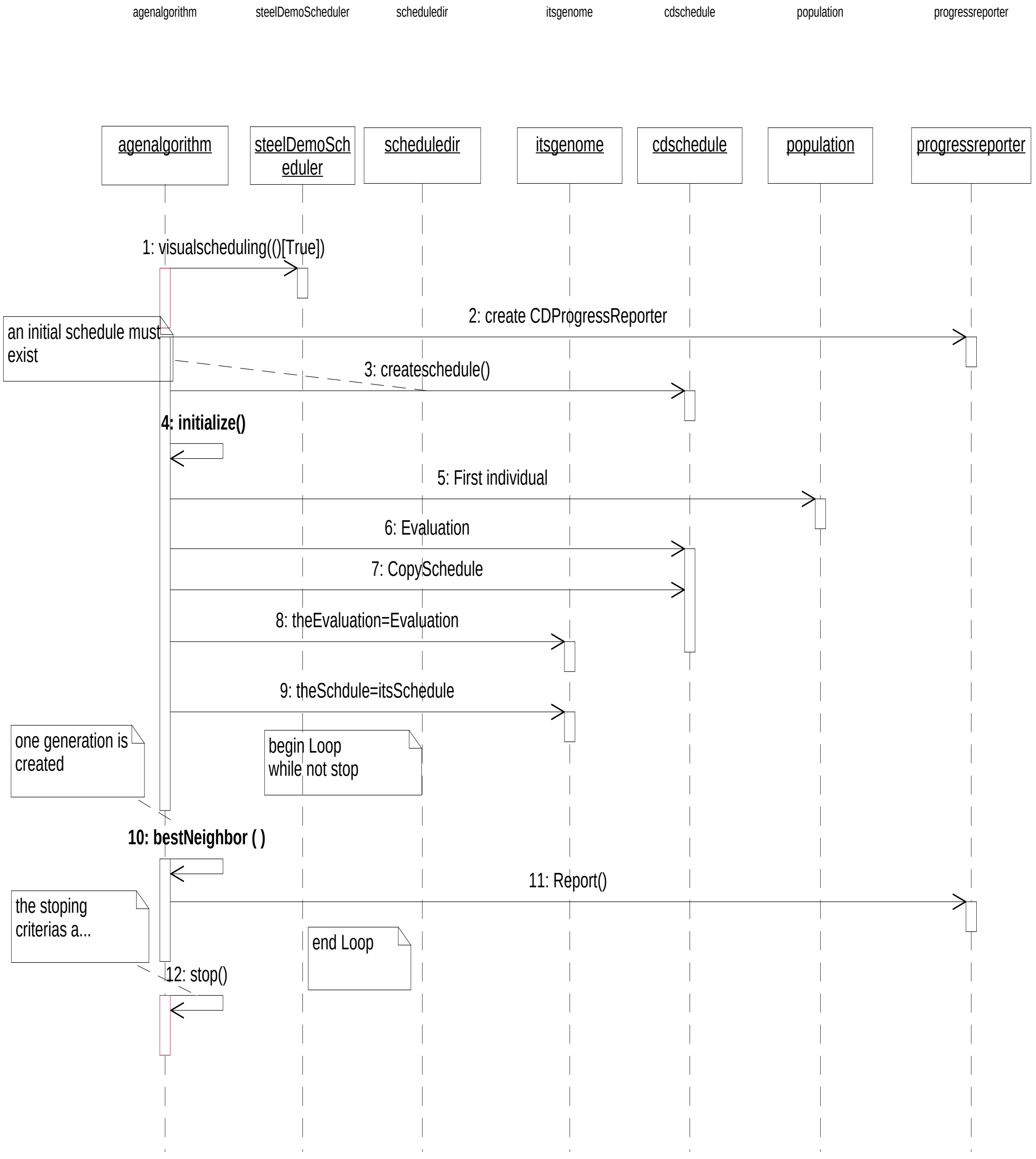


Figure 5.21: Sequence Diagram of the Improve method

Genome Class

A class **Genome** is a parametrized class, (see figure 5.24), which is a subclass of the class **GA1DArrayGenome** which is part of the **GALib Library**. It **Genome** contains the follwing methods:

- *CodedToGeneome()*
  In this case a schedule is coded to a genome. In *steeldemo application* the schedule is constructed with a set of orders, each order is coded to an integer which represents a gene of the genome. The genome in this application is a one dimentional array in which each element corresponds to an order. There are some cases where two orders must be scheduled after each other, in this case the two orders represent one gene in a the corresponding genome, (see figure 5.22).

- *Decodegenome()*
  To decode a genome to schedule, each genome is decoded to an order list, then this order list is scheduled with the AppendOrder(Order) method from the super class **CDScheduleDirector** that exists already in the *DéjàVu class framework*, in this case the old schedule is deleted and a new one is created. After that the schedule corresponding to a genome is evaluated.The evaluation function is assigned to the fitness of the genome, (see figure 5.23). To show how the method *Decodegenome()* works a sequence diagram is presented, (see figure 5.25).

- *GetGenomeSize()*
  In this method a genome is given a size.

- *AssignEvalToGenome()*
  After determining the evalution of a schedule, the function is assigned to the corresponding genome as a fitness function.

- *CreateGenome()*
  A new genome is created from a list of orders.

- *WhichCrossover(CrossoverType)*
  In this methode one crossover operator is chosen for application to genetic algorithm.

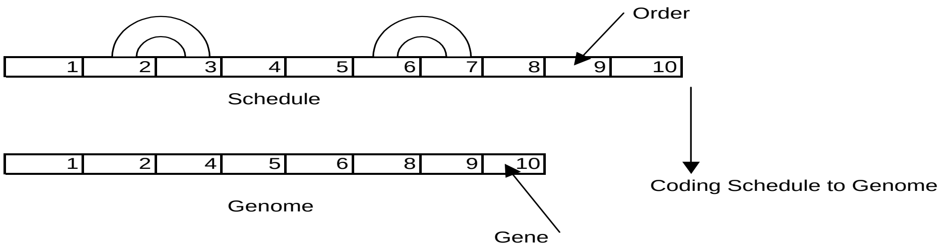


Figure 5.22: Coding Genome to a List of orders

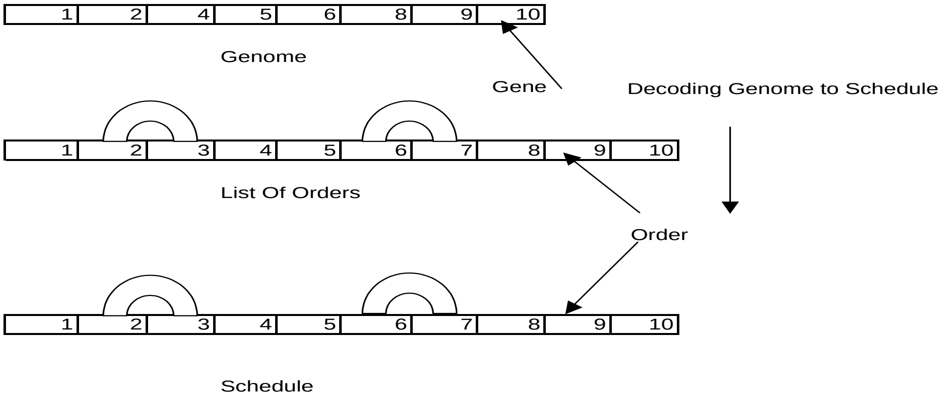


Figure 5.23: Decoding schedule to genome

- *WhichMutator(MutationType)*

  A mutation operator is chosen for the genetic algorithm.

- *clone()*

  A new genome is created from another genome.

- *copy()*

  After cloning a genome, the genome is then copied.

- *mutate()*

  this method applies a mutation operator to a genome.

- *AlternatingPosCrossover(const GAGenome ,const GAGenome ,GAGenome, GAGenome)*

  This method represents a crossover operator. (See figure 5.26)

- *EdgeRecombCrossovere(const GAGenome ,const GAGenome , GAGenome, GAGenome)*

  This method represents a crossover operator. (See figure 5.26)

- *HeuristicCrossver(const GAGenome, const GAGenome, GAGenome, GAGenome)*

  This method represents a crossover operator. (See figure 5.26)

- *OrderBasedCrossver(const GAGenome, const GAGenome, GAGenome, GAGenome)*

  This method represents a crossover operator. (See figure 5.26)

- *PositionBasesCrossover(const GAGenome, const GAGenome ,GAGenome, GAGenome)*

  This method represents a crossover operator. (See figure 5.26)

- *MaximalPreservativeCrossover(const GAGenome, const GAGenome, GAGenome, GAGenome)*

  This method represents a crossover operator. (See figure 5.26)

- *UniformaOrderCrossover(const GAGenome, const GAGenome ,GAGenome, GAGenome)*

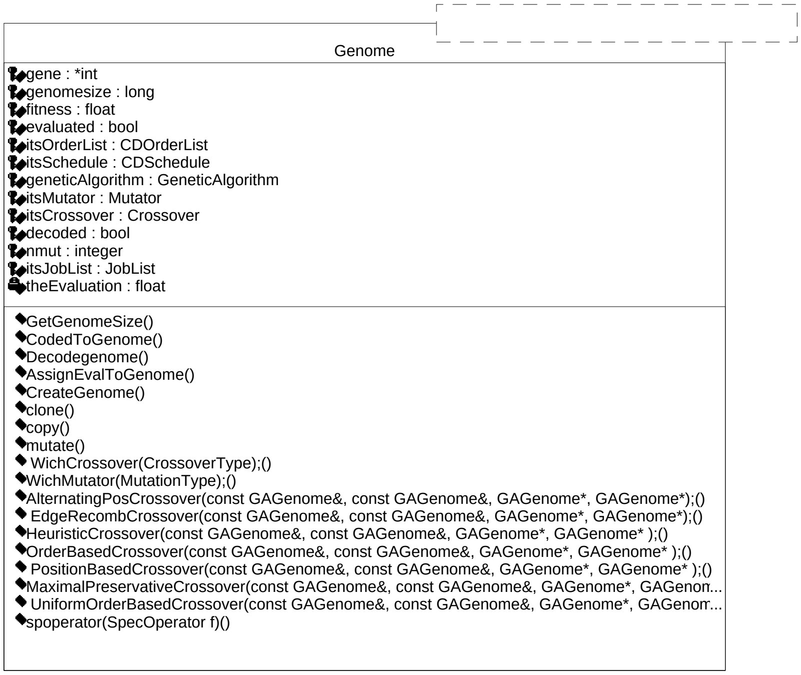

Genome
gene : *int
genomesize : long
fitness : float
evaluated : bool
itsOrderList : CDOrderList
itsSchedule : CDSchedule
geneticAlgorithm : GeneticAlgorithm
itsMutator : Mutator
itsCrossover : Crossover
decoded : bool
nmut : integer
itsJobList : JobList
theEvaluation : float
GetGenomeSize()
CodedToGenome()
Decodegenome()
AssignEvalToGenome()
CreateGenome()
clone()
copy()
mutate()
WichCrossover(CrossoverType);()
WichMutator(MutationType);()
AlternatingPosCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);()
EdgeRecombCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);()
HeuristicCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);()
OrderBasedCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);()
PositionBasedCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);()
MaximalPreservativeCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenom...
UniformOrderBasedCrossover(const GAGenome&, const GAGenome&, GAGenome*, GAGenom...
spoperator(SpecOperator f)()

Figure 5.24: Genome Class and its Interfaces

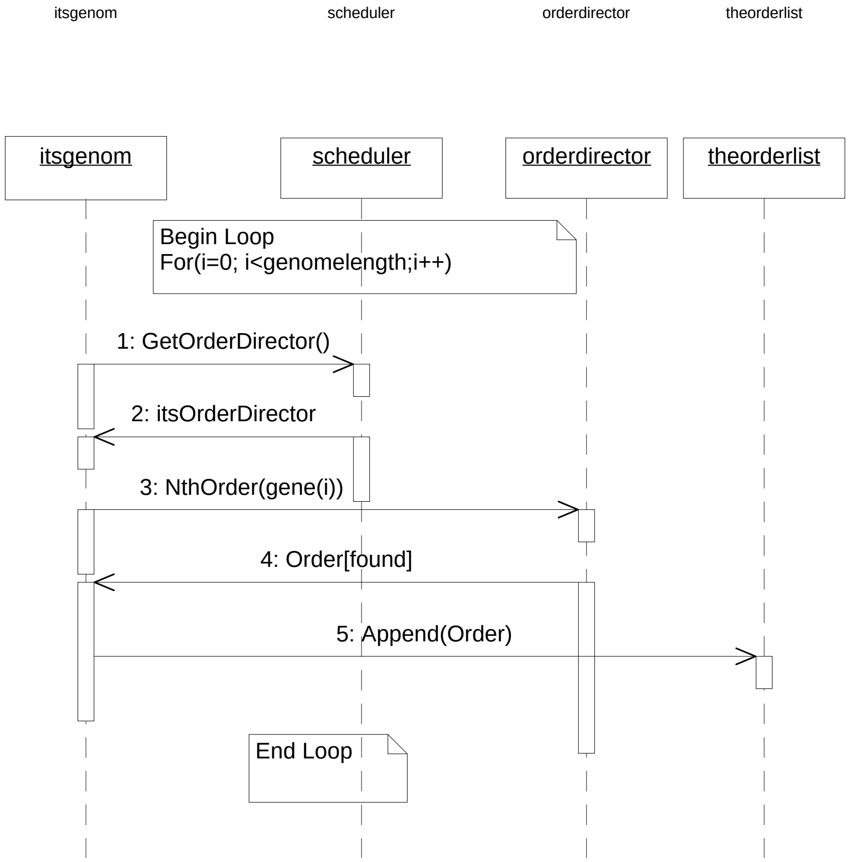


Figure 5.25: Sequence Diagram of the Decode Genome method

Operators

The operators are considered as tasks, this means that the classes consisting operators are derived from **Task** class which exists in the *DéjàVu framework*. Figures 5.26 and 5.27 succesivelly represent a class diagram and a sequence diagram of the operators.

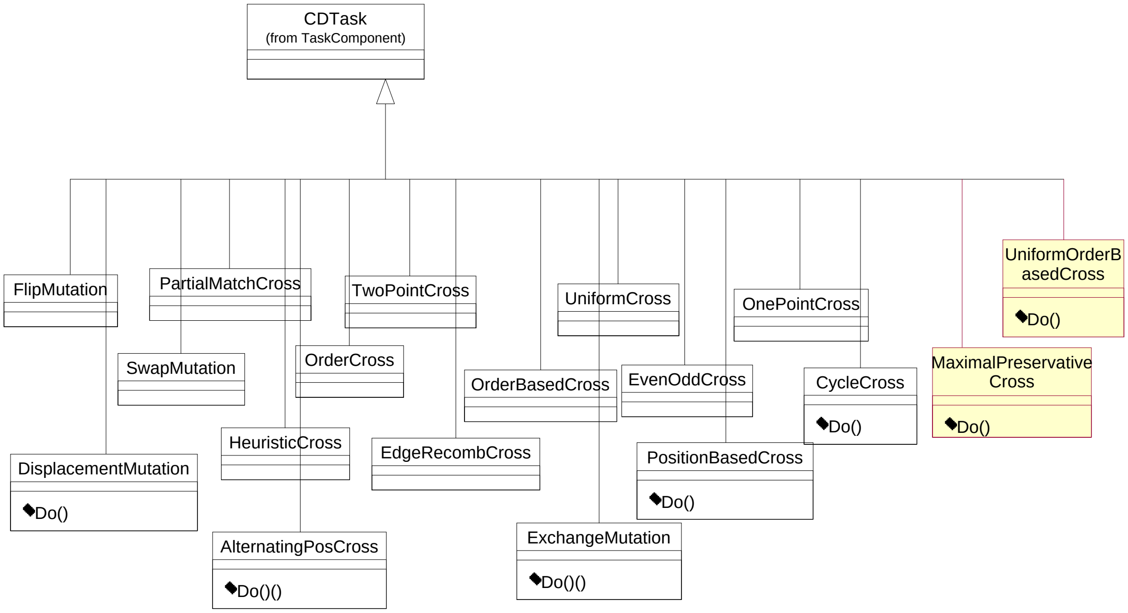


Figure 5.26: Class Diagram: Operators

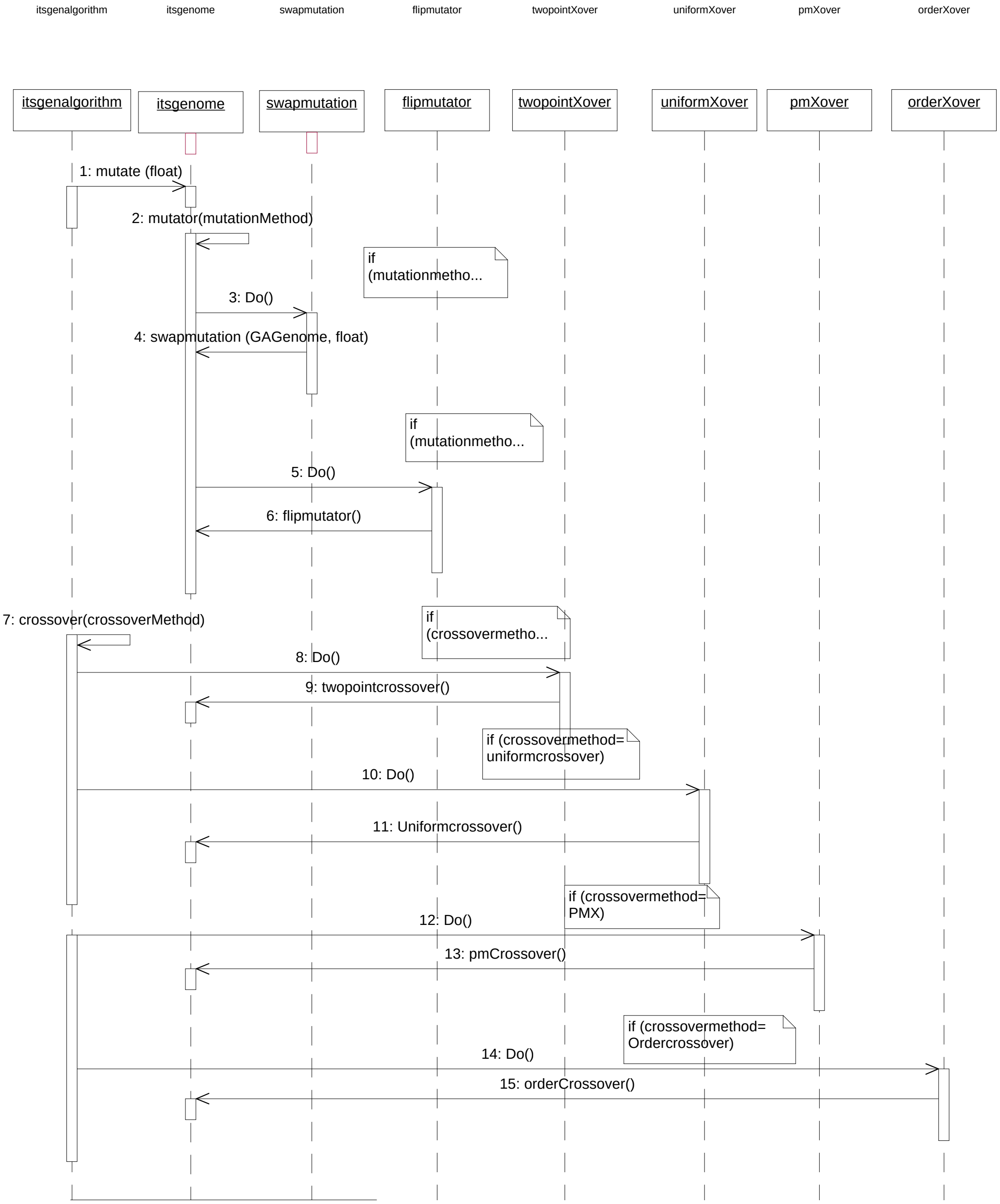


Figure 5.27: Sequence Diagram: Operators

Other Diagrams

The follwing, a class diagram, represents the design of *steady state genetic algorithm* in the *DéjàVu framework*, (see figure 5.28). In figure 5.29 a class diagram of the optimization is represented. Finally, in figure 5.30 a sequence diagram is shown.

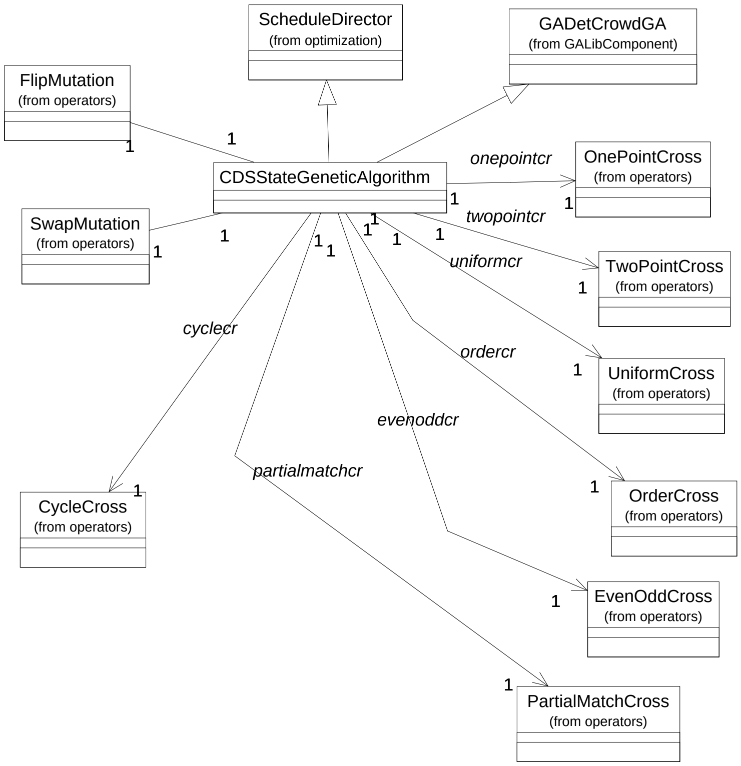


Figure 5.28: Class Diagram: Steady State Genetic Algorithm

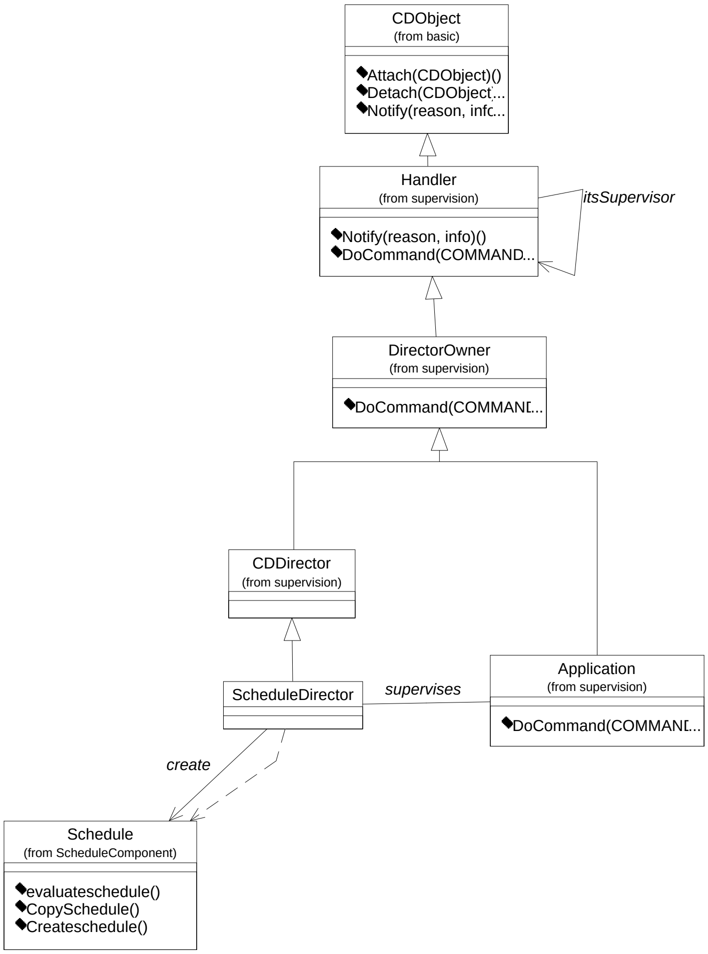


Figure 5.29: Class Diagram: Optimization

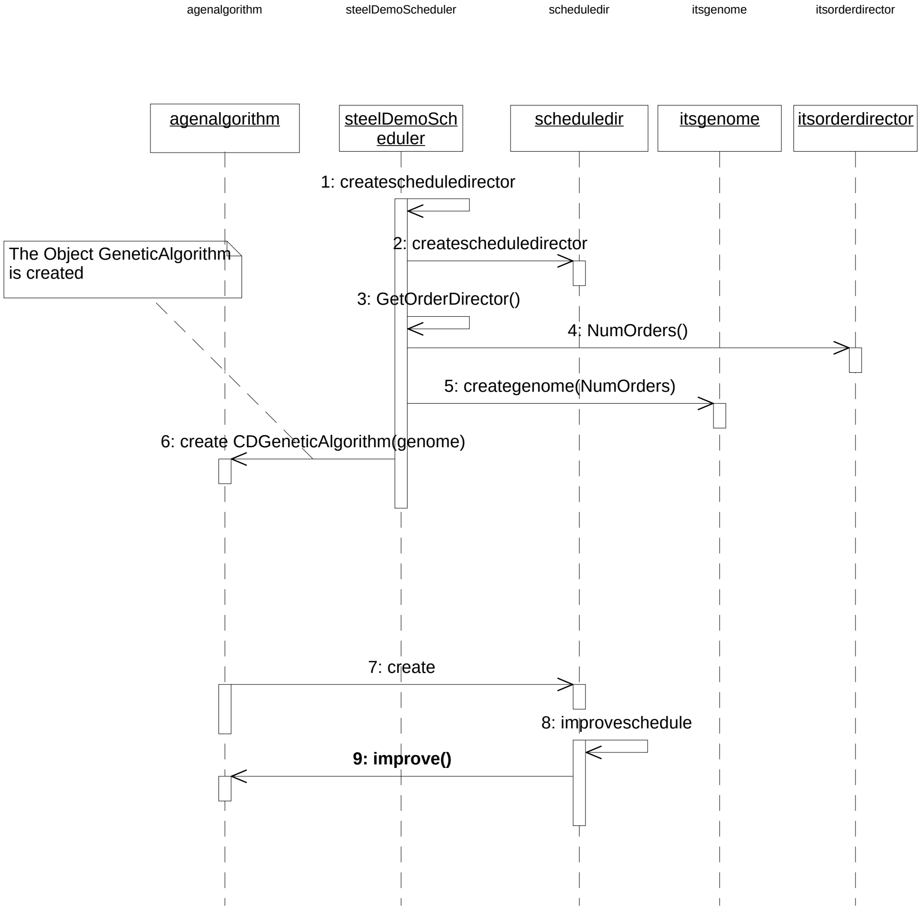


Figure 5.30: Sequence Diagram of the CreateScheduleDirector method

Chapter 6

Experiments and Results

6.1 Introduction

In this chapter, experimentations with genetic algorithms and tabu search are presented. The experiments are done on a 450 PC using the Windows 98 system. This chapter will show how the genetic algorithms parameters, such as crossover and mutation operators influence the performance of the solution. Two kind of experiments will be presented, the first one is about the operators of genetic algorithms and the second one is done to compare the improvement of the solution between genetic algorithms and the tabu search method. Some genetic algorithms are applied to the benchmarks of the steel demo application in chapter 2, and are compared with the tabu search method. Tables and graphs are presented to illustrate these experiments.

6.2 Application: Steel Demo Scheduler

Most of the experiments are done with *steady state genetic algorithm* and *simple genetic algorithm* as described by Goldberg [93]. In the first phase of the experiments, a combination of some crossover and mutation operators are made to see how the genetic operators influence the performance of genetic algorithms to improve the solution. In the second phase, steady state genetic algorithm and simple genetic algorithm are compared to the tabu search method. This phase uses the steel demo application existing in the *Déjàvu* Class library, this application is explained in details in chapters 2 and 5.

Ten different benchmarks from the steel demo application are used to compare the tabu search method with other genetic algorithms. For each experiment with genetic algorithms there are ten runs through executed. The next two sections present in detail the two phases.

6.2.1 The influence of Genetic Operators

In the first phase some crossover and mutation operators are combined to see how they influence the performance of the genetic algorithm. The genetic algorithm used in this phase is a *steady state genetic algorithm.*

The crossover operators used are: *maximal preservative crossover, order based crossover, order crossover, partial matched crossover, position based crossover,and uniform order based crossover.* Each crossover operator mentioned above is combined with the following mutation operators: *displacement mutation, scramble mutation, insertion mutation, swap mutation, and inversion mutation.* In order to see the influence of the combination of crossover and mutation operators, the same parameters for all the experiments with *steady state genetic algorithm* are used. The *steady state genetic algorithm* parameters are presented in table 6.1.

Genetic Algorithm Parameters	**Values**
Initial Population	Random
Population Size	50
Elapsed Time(s)	300
Selector	Roulette Wheel Selector
Number Of Selection	10
Crossover Probability	0.99
Mutation Probability	0.01
Scheduling Problem	BM9

Table 6.1: Parameters for the Steady State Genetic Algorithm

After ten runs for each combination of crossover and mutation operators have been completed, the results are presented in table 6.2. This table shows the best, average, and the worst results using the *steady state genetic algorithm* applied with a combination of the genetic algorithm operators. The best results are obtained with the combination of the following crossover and mutation operators.

- Maximal preservative crossover with swap mutation, insertion mutation, and scramble mutation.

- Partial matched crossover with swap mutation, insertion mutation, and scramble mutation.

- Uniformed based crossover with insertion mutation, swap mutation, and scramble mutation.

- Position based crossover with insertion mutation, scramble mutation, and swap mutation.

- Order crossover with insertion mutation, scramble mutation, swap mutation.

- Order based crossover with scramble mutation, insertion mutation, swap mutation.

The average results shows, the improvement of the solution is reached with the following combination of crossover and mutation operators:

- Partial matched crossover with swap mutation, insertion mutation, scramble mutation.

- Maximal preservative crossover with scramble mutation, insertion mutation, swap mutation.
- Order crossover with scramble mutation, insertion mutation, swap mutation.
- Order based crossover with insertion mutation, swap mutation, scramble mutation.
- Uniform order based crossover with insertion mutation, swap mutation, scramble mutation.
- Position based crossover with insertion mutation, scramble mutation, swap mutation.

Operators	**Swap Mutation**	**Scramble Mutation**	**Insertion Mutation**	**Inversion Mutation**	**Displacement Mutation**
PMX(Number of Runs)	10	10	10	10	10
worst results	0,8137	0,8004	0,8074	0,7761	0,7878
average results	0,8270	0,8103	0,8198	0,7880	0,7965
best results	***0,8454***	**0,8192**	**0,8320**	**0,8004**	**0,8036**
OrderBasedX(Number of Runs)	10	10	10	10	10
worst results	0,7902	0,7830	0,7847	0,7699	0,7704
average results	0,8047	0,8015	0,8160	0,7771	0,7785
best results	**0,8176**	**0,8264**	**0,8229**	**0,7909**	**0,7972**
OrderX(Number of Runs)	10	10	10	10	10
worst results	0,7988	0,7957	0,7862	0,7840	0,7917
average results	0,8052	0,8108	0,8096	0,7923	0,7995
best results	**0,8201**	**0,8262**	**0,8309**	**0,8043**	**0,8082**
MaximalPreservativeX(Number of Runs)	10	10	10	10	10
worst results	0,8028	0,8059	0,8075	0,7878	0,7965
average results	0,8159	0,8232	0,8216	0,7970	0,8066
best results	***0,8469***	**0,8329**	**0,8356**	**0,8020**	**0,8149**
UniformOrderBasedX(Number of Runs)	10	10	10	10	10
worst results	0,7925	0,7838	0,7949	0,7633	0,7714
average results	0,8047	0,7980	0,8148	0,7745	0,7780
best results	**0,8327**	**0,8099**	**0,8367**	***0,7807***	**0,7840**
PositionBasedX (NumberOfRuns)	10	10	10	10	10
worst results	0,7864	0,7901	0,7838	0,7602	0,7728
average results	0.7989	0,8044	0,8057	0,7729	0,7786
best results	**0,8145**	**0,8309**	**0,8359**	***0,7807***	**0,7926**

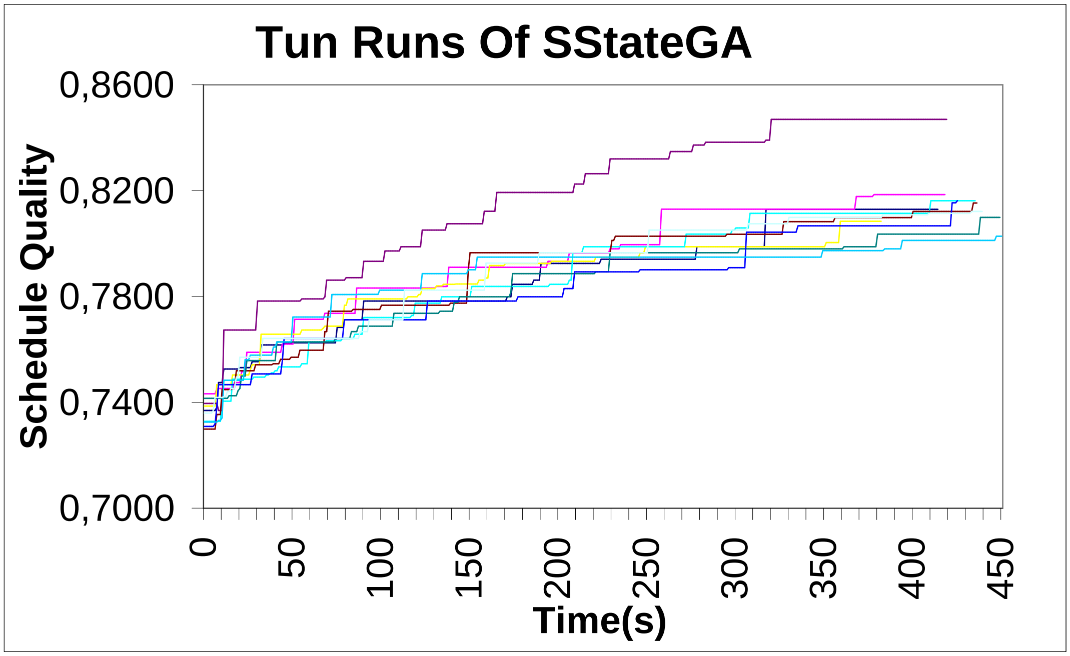
Table 6.2: Steady State Genetic Algorithm applied to the problem BM9

The figures 6.1, 6.2, and 6.3 illustrate the best results obtained from the table 6.2 using ten runs of steady state genetic algorithm combining some crossover operators with some mutation operators.

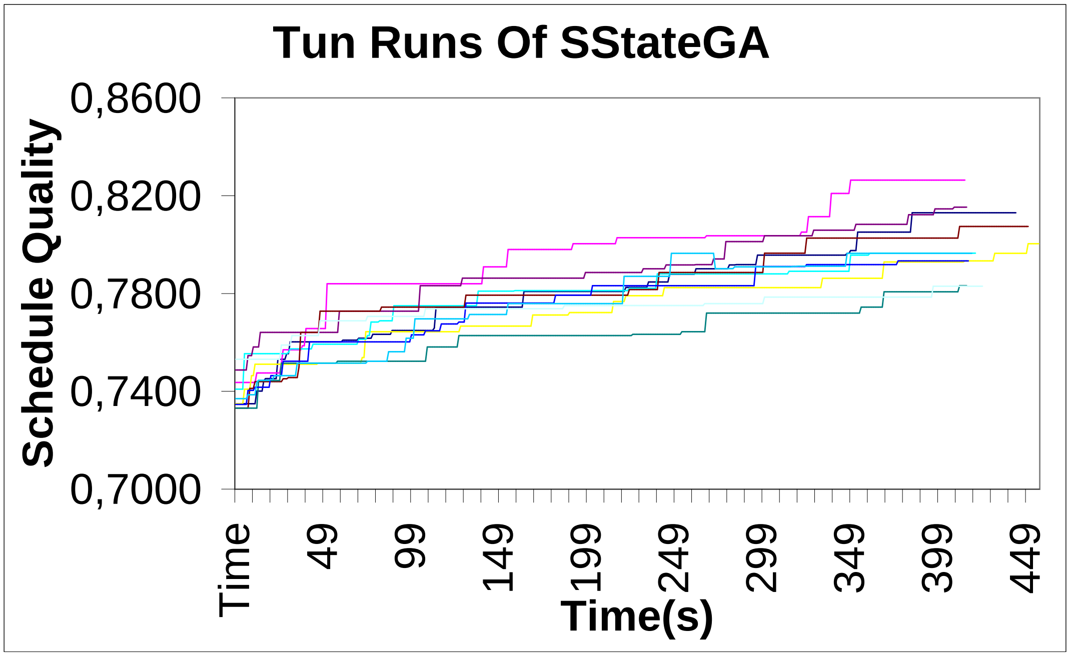
In figure 6.1 the steady state genetic algorithm is applied to the schedule of the order BM9 using *Maximal Preservative Crossover* with *Swap Mutation* and *Order Base Crossover* with *Scramble mutation.*

In figure 6.2 the steady state genetic algorithm is applied to the schedule of the order BM9 using *Order Crossover* with *Insertion mutation* and *Partial Matched Crossover* with *Swap Mutation.*

In figure 6.3 the steady state genetic algorithm is applied to the schedule of the order BM9 using *Position based crossover* with *Insertion mutation* and *Uniform order base crossover* with *Insertion mutation.*

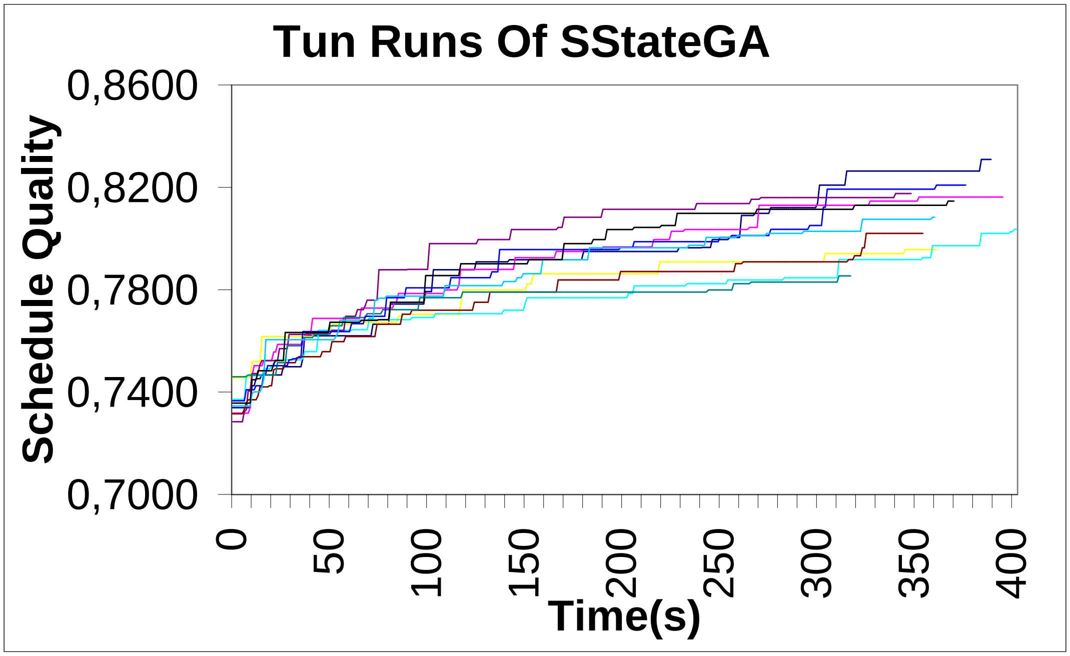


(a)

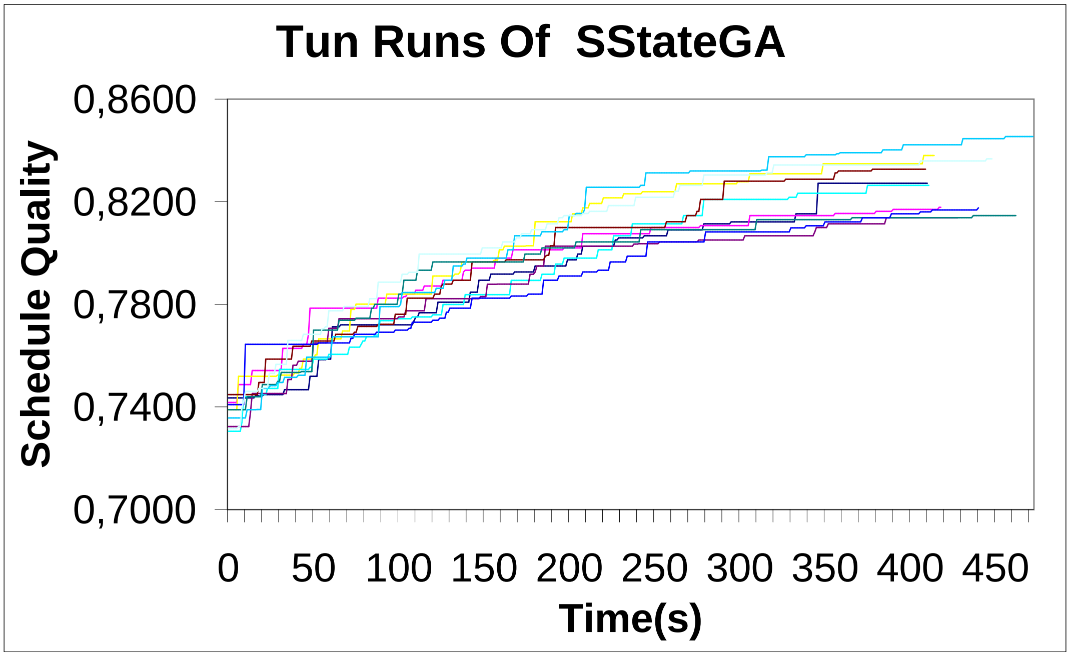


(b)

Figure 6.1: (a) Maximal Preservative Crossover with Swap Mutation. (b) Order Base Crossover with Scramble Mutation

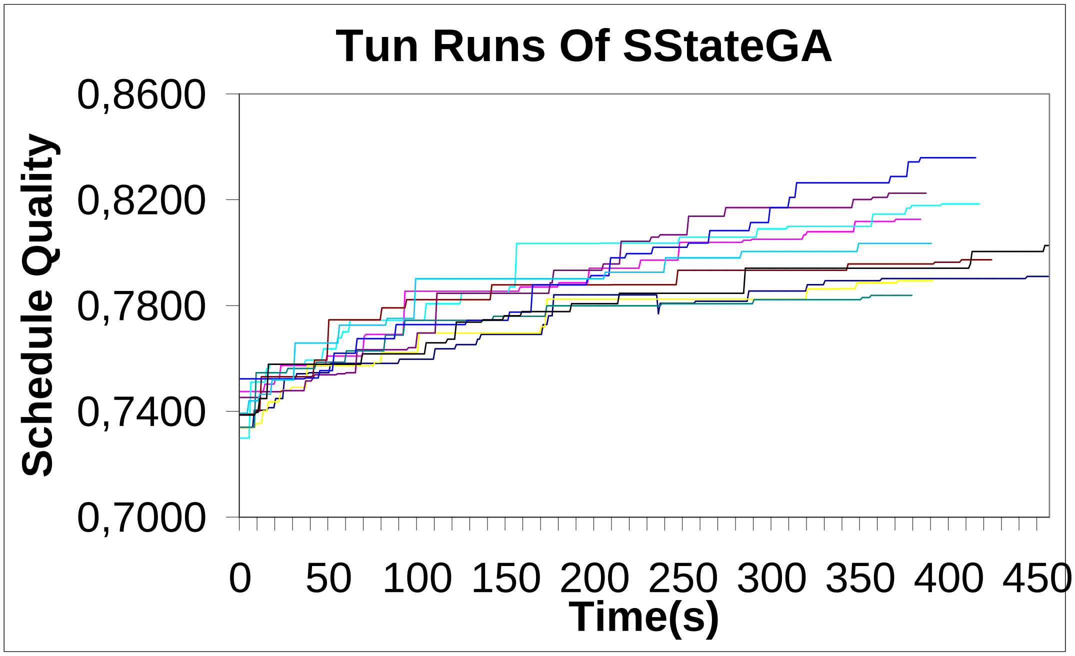


(c)

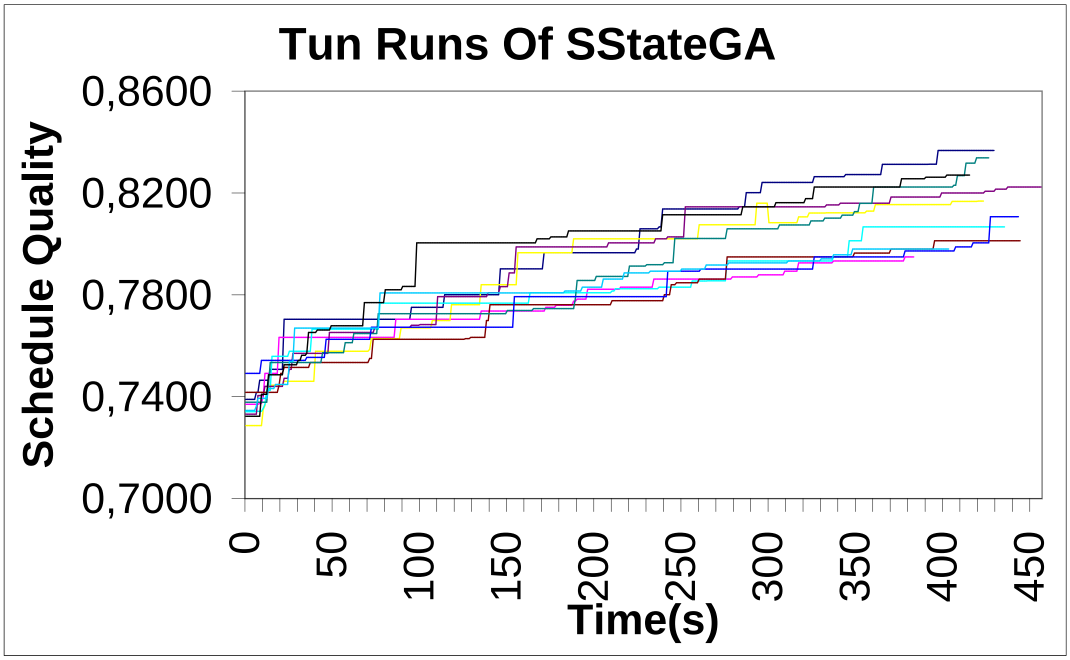


(d)

Figure 6.2: (c) Order Crossover with Insertion Mutation. (d) Partial Matched Crossover with Swap Mutation



(e)



(f)

Figure 6.3: (e) Position Based Crossover with Insertion Mutation. (f) Uniform Order Base Crossover with Insertion Mutation

6.2.2 Comparison of genetic algorithms with tabu search method

The second phase of these experiments compares two genetic algorithms with the tabu search method. The genetic algorithms used are *steady state genetic algorithm* and *simple genetic algorithm* [93]. For each benchmark BMi, $i = 1$ to 10 the parameters of the *simple genetic algorithm* are presented in table 6.3 and the parameters of the *steady state genetic algorithm* are presented in table 6.4.

Genetic Algorithm Parameters	**Values**
Initial Population)	Not Random
Population Size	50
Elapsed Time(s)	300
Selector	Roulette Wheel Selector
Crossover Probability	0.99
Mutation Probability	0.01

Table 6.3: Parameters for the Simple Genetic Algorithm

Genetic Algorithm Parameters	**Values**
Initial Population)	Not Random
Population Size	50
Elapsed Time(s)	300
Selector	Roulette Wheel Selector
Number Of Selection	10
Crossover Probability	0.99
Mutation Probability	0.01

Table 6.4: Parameters for the Steady State Genetic Algorithm

In figure 6.5 (a): The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Position Based Crossover
- *Mutation Operator:* Scramble Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Position Based Crossover
- *Mutation Operator:* Scramble Mutation

In figure 6.5 (b) The Genetic operators for simple Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

In figure 6.6 (c) The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

In figure 6.6 (d), The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Insertion Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Insertion Mutation

In figure 6.7 (e), The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

In figure 6.7 (f), The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Order Based Crossover
- *Mutation Operator:* Insertion Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

In figure 6.8 (g), The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Position Based Crossover
- *Mutation Operator:* Scramble Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Maximal Preservative Crossover
- *Mutation Operator:* Swap Mutation

In figure 6.8 (h), The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Position Based Crossover
- *Mutation Operator:* Scramble Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Position Based Crossover
- *Mutation Operator:* Scramble Mutation

In figure 6.9 (i), The genetic operators for Simple Genetic Algorithm:

- *Crossover Operator:* Partial Matched Crossover
- *Mutation Operator:* Swap Mutation

The genetic operators for Steady State Genetic Algorithm:

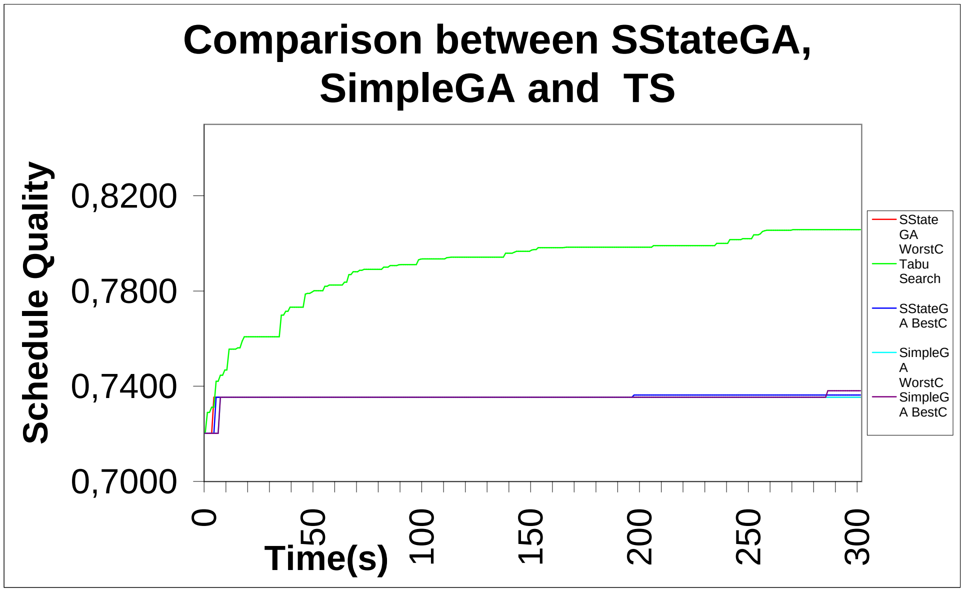
- *Crossover Operator:* Partial Matched Crossover
- *Mutation Operator:* Swap Mutation

In figure 6.9 (j), The genetic operators for Simple Genetic Algorithm:

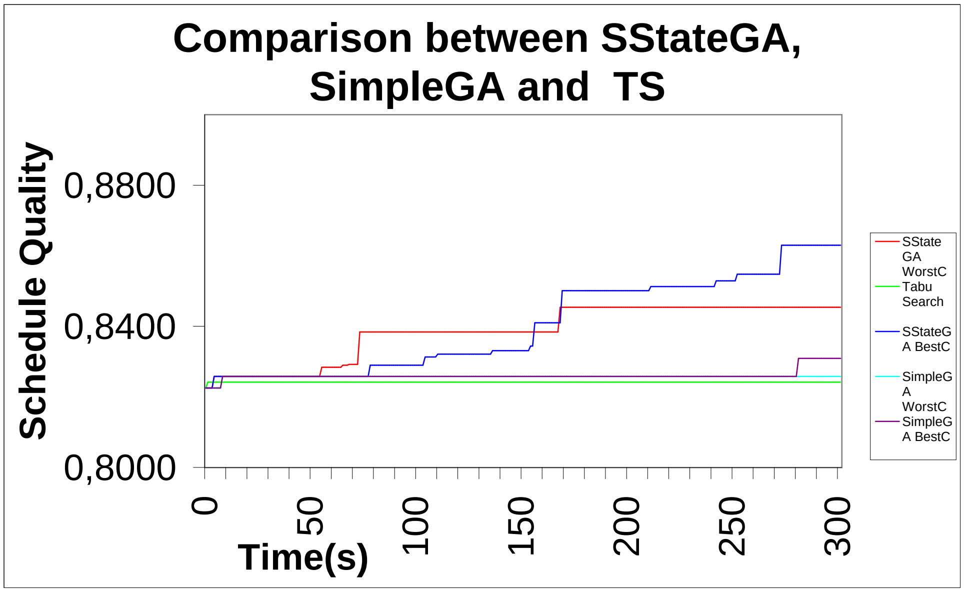
- *Crossover Operator:* Order Crossover
- *Mutation Operator:* Swap Mutation

The genetic operators for Steady State Genetic Algorithm:

- *Crossover Operator:* Order Crossover
- *Mutation Operator:* Swap Mutation



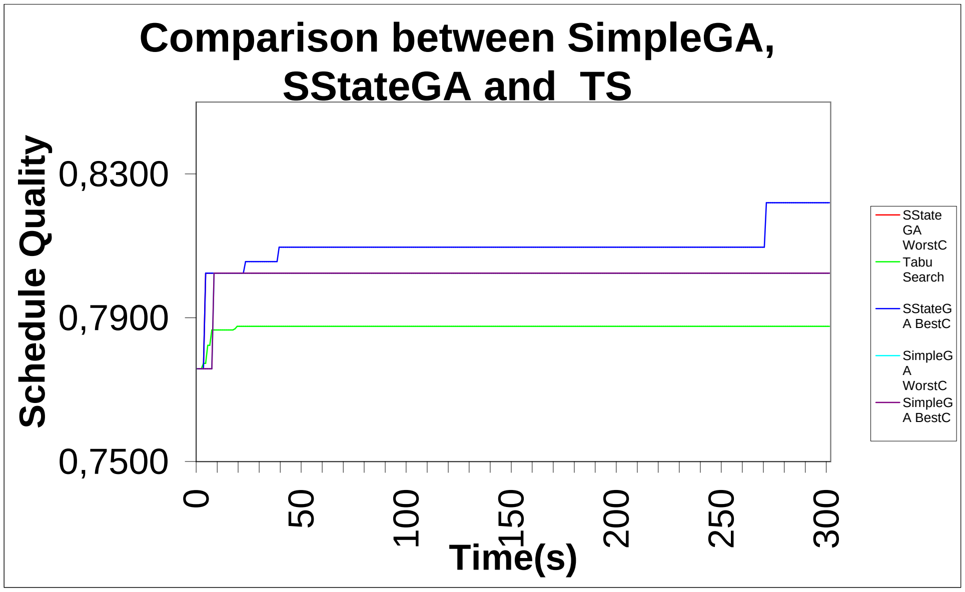
(a)



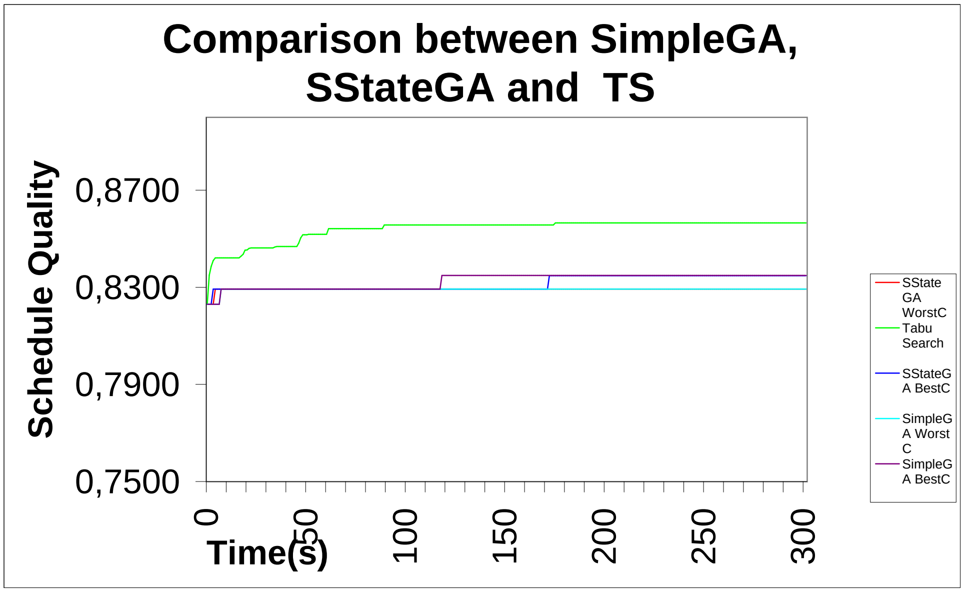
(b)

Figure 6.4: (a) Comparison of SimpleGA, SStateGA, and Tabu Search:BM6

Figure 6.5: (b) Comparison of SimpleGA, SStateGA, and Tabu Search:BM9

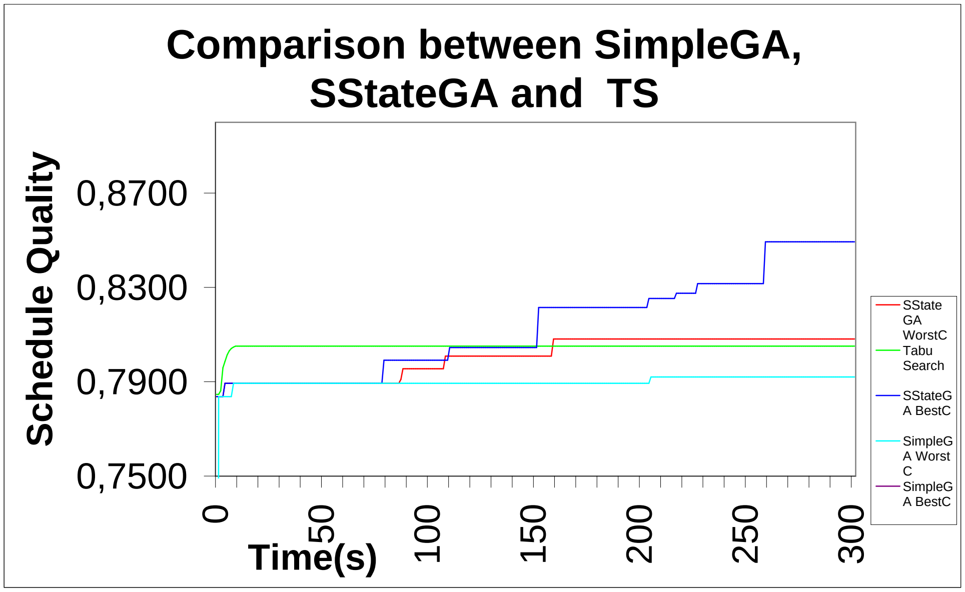


(c)

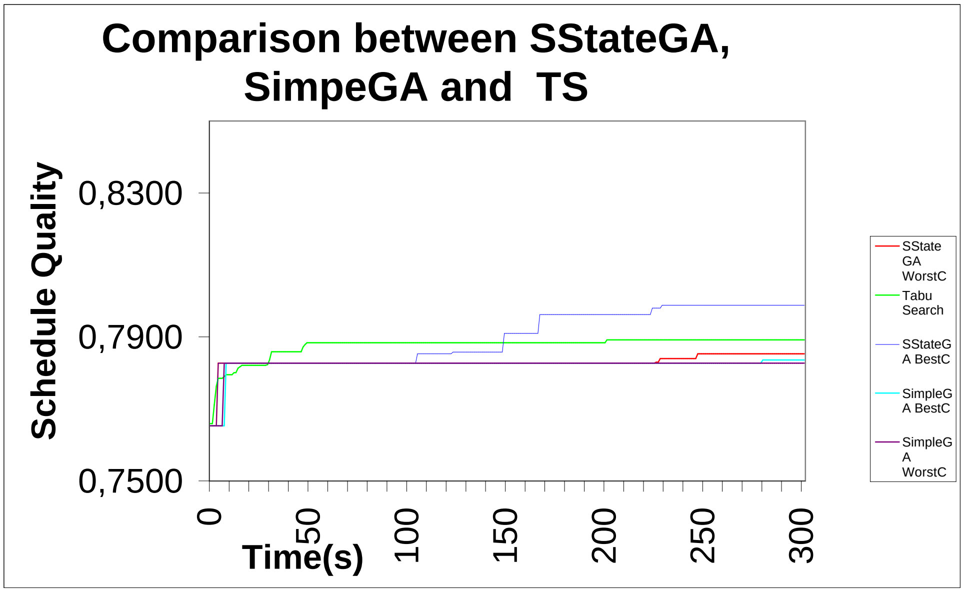


(d)

Figure 6.6: (c) Comparison of SimpleGA, SStateGA, and Tabu Search:BM10 (d) Comparison of SimpleGA, SStateGA, and Tabu Search:BM8

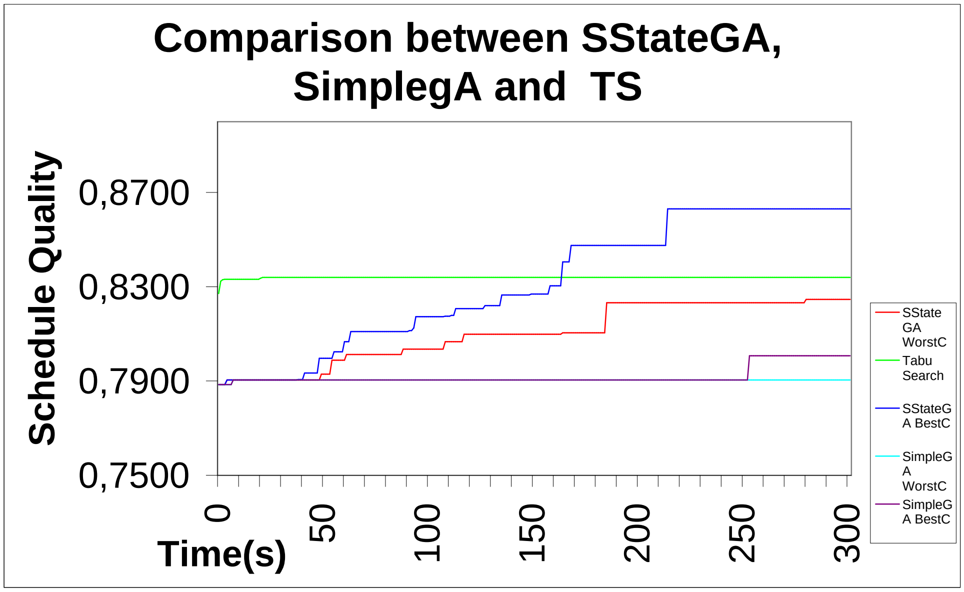


(e)

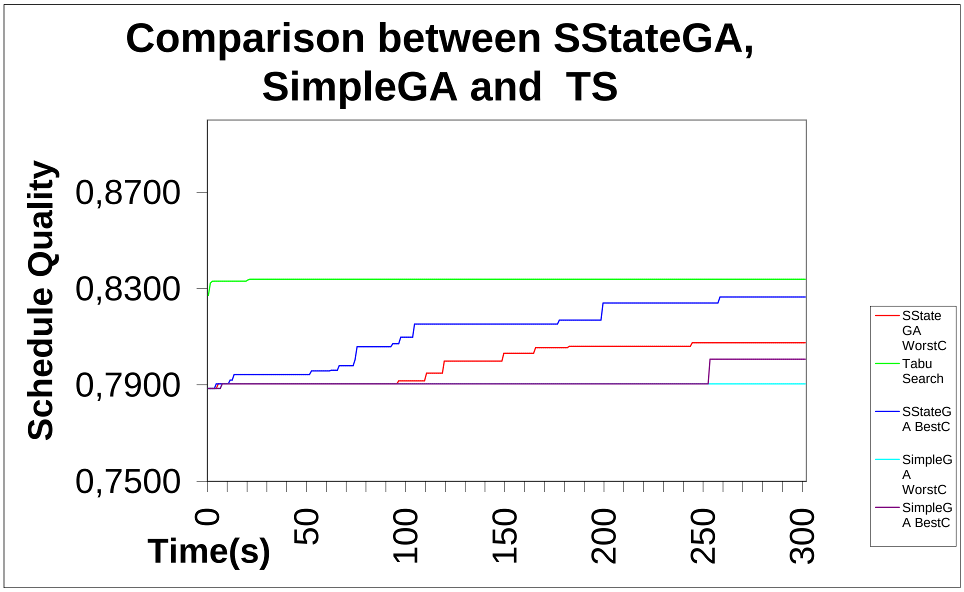


(f)

Figure 6.7: (e) Comparison of SimpleGA, SStateGA, and Tabu Search:BM1 (f) Comparison of SimpleGA, SStateGA, and Tabu Search:BM3

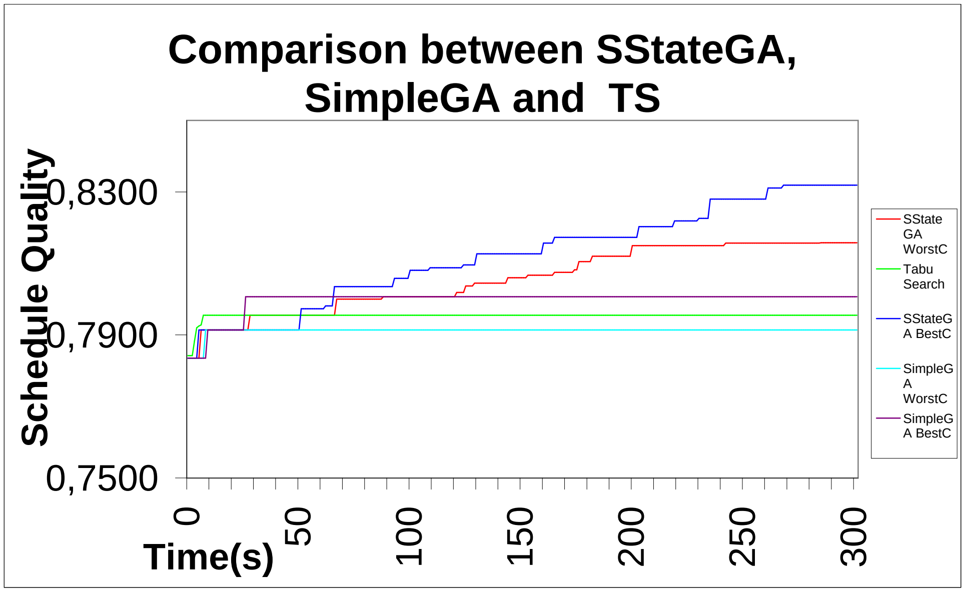


(g)

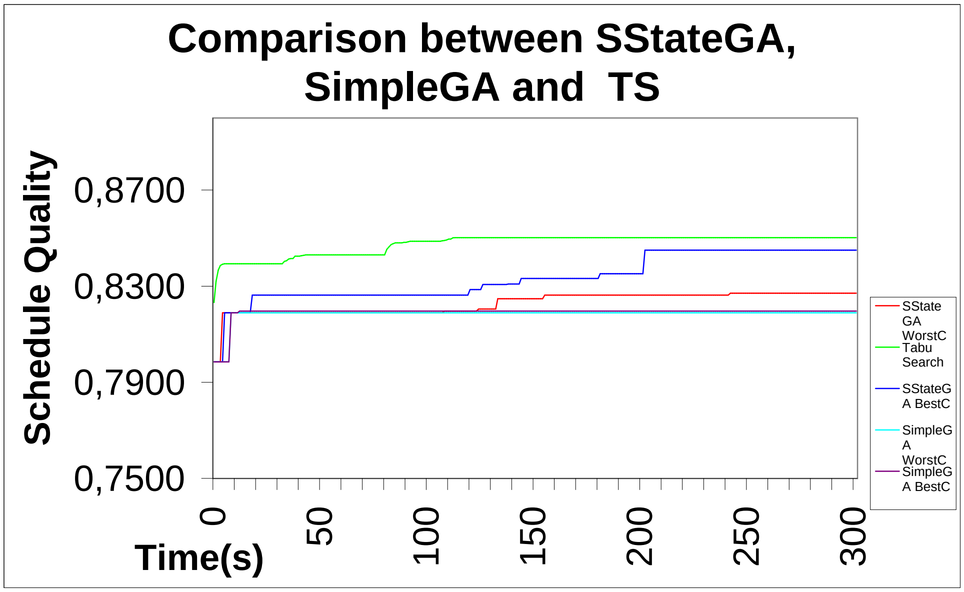


(h)

Figure 6.8: (g) Comparison of SimpleGA, SStateGA using PositionBased Crossover and Scramble Mutation , and Tabu Search:BM2 (h) Comparison of SimpleGA, SStateGA using:MaximalPreservativ Crossover and SwapMuation, and Tabu Search:BM2



(i)



(j)

Figure 6.9: (i) Comparison of SimpleGA, SStateGA using PositionBased Crossover and Scramble Mutation , and Tabu Search:BM4 (j) Comparison of SimpleGA, SStateGA using:MaximalPreservativ Crossover and SwapMuation, and Tabu Search:BM5

In figure 6.5 (a) tabu search method gives the best result. In figure 6.5 (b) *Steady State Genetic Algorithm* gives the best results, these results are better than the results with the tabu search because we used good crossover operator and mutation operator, namely maximal preservative crossover and swap mutation. In figure 6.6 (c) *Simple genetic algorithm* and *steady state genetic algorithm* gives better results than tabu search method, and in figure 6.6 (d) tabu search is better than *simple genetic algorithm and steady state genetic algorithm.* In both figures the crossover operator used was maximal preservative crossover but they used different mutation operators, namely the swap mutation in figure 6.6 (c) and insertion mutation in figure 6.6 (d). These explain that genetic algorithms with swap mutation improve the solution better than using the insertion mutation. In figures 6.7 (e) and 6.7 (f) the tabu search is better than *simple genetic algorithm* but the *steady state genetic algorithm* gives better results than the tabu search method. In figure 6.8 (g) the tabu search method gives a better result than the *simple genetic algorithm* but *steady state genetic algorithm* using maximal preservative crossover and swap mutation, is better than the tabu search. In figure 6.8 (h) tabu search is better than *simple genetic algorithm* and *steady state genetic algorithm.* This means that position based crossover and scramble are not powerful enough for the genetic algorithms. In figure 6.8 (i) partial matched crossover and swap mutation are used. In this case *steady state genetic algorithm* and *simple genetic algorithm* give better results than the tabu search method. In figure 6.8 (j) where order crossover and swap mutation are used, the tabu search method improve the solution better than the improvement with *steady state genetic algorithm* and *simple genetic algorithm.* In these two cases the partial matched crossover and swap mutation perform well in the genetic algorithms

6.3 Conclusion

These Experiments show that genetic operators play an important role in improving solution in genetic algorithms. The experiments also show that genetic algorithms sometimes give better solutions than the tabu search method using the appropriate crossover and mutation operators. We can say that the use of maximal preservative crossover combined with swap mutation and partial matched crossover combined with swap mutation in genetic algorithms gives better results than the tabu search method. The one inconvenience of genetic algorithms is that they are slower to reach good results than the tabu search method because each genome in the population has to be decoded to a schedule.

Chapter 7

Epilogue

In this chapter a general conclusion is provided as well as presenting open topics of work to be done in the future. We illustrate the advantages and disadvantages of the use of genetic algorithms.

7.1 General Conclusion

Genetic algorithms used in this thesis have shown how they can give better solution than the tabu search method, if we use the appropriate genetic operators like crossover and mutation operators. If the initial population in the genetic algorithms is not randomly created, but is created from an initial schedule then the genetic algorithms converge to make a good solution. In this thesis we did not experiment with other kinds of chromosome representations but we worked on one presentation, which is an array of numbers, because this representation is closer to the natural representation of a schedule. In this thesis the objective function is designed as a single function and this is why we did not used multi-objective genetic algorithms, which are used in many applications [157], [79], [213], [19], [153], [216], [29],[36], [12], [125], [56], [214], [4], [116], [142], [86], [188], [112], [13].

7.2 Open Topics

- How can genetic algorithms work faster?

  In this thesis we used simple genetic algorithm and steady state genetic algorithms to improve the solution. The results were good but the algorithms did not work very fast. In the future we can use parallel genetic algorithms [39],[113] where genetic algorithms work on many sub populations at the same time.

- Design of theoretical classes of algorithms using the *strategy design pattern*
  In *DéjàVu Class framework* there is no theoretical classes for algorithms. In the future we will design theoretical classes which will represent improved algorithms using the *strategy design pattern* [89].

- The design of a new multi-objective function where we can use multi-objective genetic algorithms.

- How we use methods from machine learning to solve the steel making scheduling problem? The system should learn how to solve the scheduling problem using evolutionary algorithms such as genetic algorithms to give us "good" solution.

Bibliography

[1] Jesús S. Aguilar-Ruiz, Isabel Ramos, José C. Riquelme, and Miguel Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43:875–882, December 2001.

[2] H. Wright Alden. Genetic algorithms for real parameter optimization. *First workshop on the fundations of Genetic Algorithms and Classifier Systems*, 1990.

[3] Shahid Ali, Sadiq M. Sait, and Muhammed S.T. Benten. Gsa: Scheduling and allocation using genetic algorithm. *Proceedings of the Conference on European Design Automation. Los Alamitod, CA:IEEE Computer Society Press.*, pages 84–89, 1994.

[4] Rodrigues Almeida Mayron, Silvio Hamacher, Marco Pacheco, Marco Aurélio, and Marley B.R. Vellasco. The energy minimization method: A multiobjective fitness evaluation technique and its application to the production scheduling in a petroleum refinery. *Evolutionary Computation*, 2001.

[5] Mikhail J. Atallah. *Algorithms and Theory of Computation Handbook.* Library of Congress Cataloging in Publication Data, 1998.

[6] Vincent Bachelet and El-Ghazali Talbi. A parallel co-evolutionary metaheuristic. *IPDPS 2000 Workshops, LNCS 1800*, pages 628–635, 2000.

[7] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1996.

[8] Thomas Bäck and Hans-Paul Hamel, Ulrich an Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions On Evolutionary Computation*, 1(1):3–17, April 1997.

[9] S. Bagchi, S. Uckum, Y. Miyabe, and K. Kawamura. Exploring problem-specific recombination operators for job shop scheduling. *International Conference Of Genetic Algorithms. ICGA*, 1991.

[10] S. Bagchi, S. Uckum, Y. Miyabe, and K. Kawamura. Exploring problem-specific recombination operators for job shop scheduling. *IEEE Expert.*, 8(5):15–24, October 1993.

[11] Tapan P. Bagchi. Pareto-optimal solutions for multi-objective production scheduling problems. *First International Conference on evolutionary multi-criterion optimization, EMO 2001*, 1993:458–471, 2001.

[12] Jerzy Balicki and Zygmunt Kitowski. Multicriteria evolutionary algorithm with tabu search for task assignment. *First International Conference on Evolutionary Multi-Criteria Optimization.Proceedings*, 1993:373–384, March 2001.

[13] Matthieu Basseur, Franck Seynhaeve, and Talbi El-ghazali. Design of multi-objective evolutionary algorithms: Application of the flow-shop scheduling problem. *Conference Proccedings on Evolutionary Computation. CEC02*, 2002.

[14] Philippe Baudet, Catherie Azzaro-Pantel, Luc Pibouleau, and Serge Domenech. Un couplage entre un algorithm génétique en un modèle de simulation pour l'ordonnancement a court terme d'un atelier discontinu de chimie fini. *RAIRO Operation Research*, 33:299–338, 1999.

[15] J.C. Bean. Genetics and random keys for sequences and optimization. Technical report, Department of Industrial and operations Engineering, The university of Michigan, Ann Arbor, 1993.

[16] J.C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[17] Adam Prügel Bennett. Modeling crossover-included linkage in genetic algorithms. *IEEE TRANSACTION ON EVOLUTIONARY COMPUTATION*, 5(4):376–387, AUGUST 2001.

[18] Peter et al. Bentley. *Evolutionary Design by Computers.* Morgan Kaufmann, May 1999.

[19] P.J. Bentley and J.P Wakefield. Finding Acceptable Solutions in the pareto-optimal range using multi-objective genetic algorithms. *Conference on soft Computing in engineering Design and manufacturing (WSC2)*, pages 23–27, June 1997.

[20] Meyer Bertrand. *Reusable Software.* Prentice-Hall object-oriented. Englewood Cliffs NJ., 1994.

[21] Christian Bierwirth and Dirk C. Mattfeld. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7(1):1–17, 1999.

[22] Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. On permutation representations for shceduling problems. *Proceedings of Parallel Problem Solving from Nature IV*, pages 310–318, 1996.

[23] Robert V. Binder. *Testing Object-Oriented Systems. Models, Patterns, and Tools.* Addison Wesley, 2000.

[24] Isabelle Blot-Thibaut. *Un système de Programmation Par Contraintes sur les domaines finies entiers.* PhD thesis, Faculté des Sciences et Techniques. Université de Bourgogne, June 1998.

[25] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language. User Guide.* Addison Wesley Longman, addison wesley longman, inc. edition, October 1998.

[26] Wilfried Brauer and Gerhard Weiß. Multi-machine scheduling- a multi-agent learning approach. *International Conference on Multi-Agent Systems*, pages 42–48, 1998.

[27] H.J. Bremermann. The evolution of intelligence. the vervous system as model of its environment. Technical report, Department of Mathematics, university of Wash., Saetle, 1958.

[28] David Brittain. *Optimization of the Telecommunication Access Network*. PhD thesis, University of Bristol, Faculty of Engineering, Department on Engineering Mathematics, January 1999.

[29] C. Brizuela, N. Sannomiya, and Y. Zhao. Multi-objective flow-shop: Preliminary results. *First International Conference on evolutionary multi-criterion optimization, EMO 2001*, 1993:443–457, 2001.

[30] Peter Brucker. *Scheduling algorithms*. Springer Verlag, 1995.

[31] R. Bruns. Incorporation of a knowledge-based scheduling system into genetic algorithm. *Proceedings of the "GI-Jharestagung", Karlsruhe*, 1992.

[32] R. Bruns. Direct chromosome representation and advanced genetic operators for production scheduling. *Proceedings of the fifth International Conference on Genetic Algorithms San Mateo, CA.*, 1993.

[33] E.K. Burke and A.J. Smith. A multi-stage approach for the thermal generator maintenance scheduling problem. *Congress on Evolutionary Computation*, 2, 1999.

[34] E.K. Burke and A.J. Smith. Hybrid evolutionary techniques for the maintennance scheduling problem. *Transaction on Power Systems*, 15(1):122–128, February 2000.

[35] Heng Cao, Haifeng Xi, Yupin Luo, Suxing Yang, and Yi Peng. Ga with hierarchical evaluation: A framework for solving complex machine scheduling problems in manufacturing. *GALESIA97. Second International Conference on Genetic Algorithms In Engineering Systems: Innovation And Applications*, (446), September 1997.

[36] W.Matthew Carlyle, Bosum Kim, John W. Fowler, and Esma S. Gel. Comparison of multiple objective gnetic algorithms for parallel machine scheduling problems. *First International Conference on evolutionary multi-criterion optimization, EMO 2001*, 1993:472–485, 2001.

[37] Hugh M. Cartwright and A. Tuson. Genetic algorithms and flow shop scheduling: towards the development of real-time process control system. *Lecture notes in Computer science 865*, pages 277–290, 1994.

[38] Kevin R. Caskey. A manufacturing problem solving environment combining evaluation, search, and generalisation methods. *computers in Industry*, 44:175–187, 2001.

[39] Y. Lee Chae and J. Soek. Kim parallel genetic algorithms for the earliness-tardiness job shop scheduling problem with penalty weights. *Computers ind. Eng.*, 30(4):231–243, 1995.

[40] N. Chaiyaratana and A.M.S. Zalzala. Recent developments in evolutionary and genetic algorithms: Theory and application. *GALESIA97. Second International Conference on Genetic Algorithms In Engineering Systems: Innovation And Applications*, (446):270–277, September 1997.

[41] Uday K. Chakraborty, D. Laha, and Mandira Chakraborty. A heuristic genetic algorithm for flowshop scheduling. . *Proceedings of the 23 rd International Conference on Information Technology Interfaces*, June 2001.

[42] W.T. Chan and H. Hu. Precast production scheduling with genetic algorithms. *Proceedings of the 2000 Congress on Evolutionary Computation*, 2:1087–1094, 2000.

[43] Stephen Chen and Stephen F. Smith. Improving genetic algorithms by search space reductions (with applications to flow shop scheduling). *GECCO-99:Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[44] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms-i. presentation. *Computers ind. Eng.*, 30(4):983–997, 1996.

[45] S.C. Chu and H.L. Fang. Genetic algorithms vs. tabu search in timetable scheduling. *Third International Conference on Kowledge-Based Intelligent Information Engineering Systems*, 1999.

[46] Gary A. Cleveland and Stephen F. Smith. Using genetic algorithms to schedule flow shop releases. *International Conference on Genetic Algorithms*, 1989.

[47] Emma Collingwood. Investigation of a multiple chromosome evolutionary algorithms for bus driver and other problems. Master's thesis, University of Edinburgh, Department of Artificial Intelligence, 1995.

[48] Arthur L. Corcoran and Roger L. Wainwright. Libga: A user-friendly workbench for order-based genetic algorithm research. *ACM/SIGAPP Symposium on Applied Computing*, pages 111–117, 1993.

[49] Carlos Cotta, Enrique Alba, and Jose M Troya. Utilizing dynastically optimal forma recombination in hybrid genetic algorithms. *Parallel Problem Solving from Nature-PPSN V. 5th International conference Amsterdam The Netherlands*, pages 305–314, September 1998.

[50] K.P. Dahal, G.M. Burt, J.R. McDonald, and A. Moyes. A case study of scheduling storage tanks using a hybrid genetic algorithm. *IEEE TRANSACTION ON EVOLUTIONARY COMPUTATION*, 5(3):283–296, June 2001.

[51] L. Davis. Applying adaptive algorithms to epistatic domains. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[52] L. Davis. Job shop scheduling with genetic algorith. *Proceedings of the First Int. Conference on Genetic Algorithms(Edited by J.Grefenstette)*, pages 136–140, 1985.

[53] L. Davis. Handbook of genetic algorithms. *New York: Van Nostrand Reinhold*, 1991.

[54] Robert A. Day. *How to Write and Publish a Scientific Paper.* Cambridge University Press, Trumpington Street, Cambridge CB2 1RP, third edition edition, 1991.

[55] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* Dissertation abstracts international, 36(10), 5140(b), University of Michigan, 1975.

[56] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable test problems for evolutionary multi-objective optimization. Technical Report 112, Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich, Gloriastrasse 35., ETH-Zentrum, CH-8002,, july 2001.

[57] Jörg Denzinger and Tim Offermann. On cooperation between evolutionary algorithms and other search paradigms. *Congress on Evolutionary Computation.*, 3, 1999.

[58] Christos Dimopoulos and Ali M. S. Zalzala. Recent develeopments in evolutionary computation for manufacturing optimization: Problems, solutions and comaparisons. *Transactions on Evolutionary Computation*, 4(2):93–113, July 2000.

[59] Jürgen Dorn. Task-oriented desing for scheduling applications. Technical report, Christian Doppler Laboratory for Expert Systems. CD-TR 93/50, Vienna University of Technology, 1993.

[60] Jürgen Dorn. Case-based reactive scheduling. Technical report, Christian Doppler Laboratory for Expert Systems. CD-TR 94/75, vienna University of Technology, Austria, 1994.

[61] Jürgen Dorn. Expert systems in the steel making industry. *Proceedings of the 2nd World Congress on Expert Systems Lisbon, Portugal*, January 1994.

[62] Jürgen Dorn. Iterative improvements methods for knowledge-based scheduling. *Artificial Intelligence Communications. AICOM*, 8(1):20–34, 1995.

[63] Jürgen Dorn. The déjàvu scheduling class library. *in Fayad, Schmidt, and Johnson (eds.) Implementation Application Frameworks, Wiley*, pages 521–540, 1999.

[64] Jürgen Dorn. Towards reusable intelligent scheduling software. *Proceedings of XPS-99: Knowledge-based systems, F.Puppe (ed), Springer Lecture Notes in Artificial Intelligence*, 1570:101–102, 1999.

[65] Jürgen Dorn and M. Girsch. Genetic operators based on constraint repair. *Proceedings of the ECAI'94. Workshop on Genetic Algorithms and other Evolutionary Algorithms, Amsterdam and is considered for publication in LNCS*, 1994.

[66] Jürgen Dorn, M. Girsch, and N. Vidakis. DÉjÀvu - a reusable framework for the construction of intelligent interactive schedulers. *Advances in Production Management Systems-Perspectives and Future Challenges-, Okino et al.(eds) Chapman and Hall*, pages 467–478, 1998.

[67] Jürgen Dorn, Mario Girsch, and Vidakis Nikos. DÈjÀ vu. a reusable framework for the construction of intelligent interactive schedulers. *In Proceedings of the International*

Conference on Advances in Production Management Systems (APMS 96), pages 637–644, November 1996.

[68] Jürgen Dorn, Mario Girsh, Günther Skele, and Wolfgang Slany. Comparison of iterative improvement techniques for scheduling optimization. *European Journal of Operational Research*, 94(2):349–361, October 1996.

[69] Jürgen Dorn and R. M. Kerr. Co-operating scheduling systems communicating through fuzzy sets. *Preprints of the 2 nd IFAC/IFIP/IFORS- Workshop on Intelligent Manufacturing Systems (IMS 94)*, pages 367–373, June 1994.

[70] Jürgen Dorn, Roger Kerr, and Gabi Thalhammer. Reactive scheduling: Improving the robustness of schedules and restricting the effects of shop floor disturbances by fuzzy reasoning. *Human-Computer Studies*, 42:687–704, 1995.

[71] Jürgen Dorn and A.D. Prianichnikova. The steeldemo scheduler. *Proceedings of the Siberian Conference on Operational Research (SCOR-98), Novosibirsk*, 1998.

[72] Jürgen Dorn and R. Shams. Scheduling high-grade steel making. *IEEE Expert*, pages 28–35, February 1996.

[73] Jürgen Dorn and Wolfgang Slany. A flow shop with compatibility constraints in a steel making plant. *Zweben and Fox (eds) Intelligent Scheduling*, pages 629–654, 1994.

[74] Jürgen Dorn, Markus Stumptner, Anna Prianichnikova, and Helmut Veith. Multiprocessor scheduling using the déjàvu scheduling class library. *Österreichische Gesellschaft für Artificial Intelligence*, 18(4):16–25, 1999.

[75] Ulrich Dorndorf and Erwin Pesch. Evolution based learning in a job shop problem. *Computers and Operations Research*, 22(1):25–40, 1995.

[76] K. Dussa-Zieger and M. Schwehm. Scheduling of parallel programs on configurable multiprocessors by genetic algorithms. *International Journal of Approximate Reasoning*, 19:23–38, 1998.

[77] A.E. Eiben and M. Schoenauer. Evolutionary computing. *Information Processing Letters*, 82(1):1–6, April 2002.

[78] H-Lan Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. *Proceedings of the fifth International Conference on Genetic Algorithms*, pages 375–382, 1993.

[79] M.P. Fanti, B. Maione, D. Naso, and B. Turchiano. Genetic multi-criteria approach to flexible line scheduling. *International Journal of Approximate Reasoning*, 19:5–21, 1998.

[80] Mohamed E. Fayad, Douglas C. Schmidt, and Ralph E. Johnson. *Building Application Frameworks. Object-Oriented foundations of Framework Design.* Wiley Computer Publishing, 1999.

[81] Paul Field. *A multary Theory for Genetic Algorithms: Unifying Binary and Nonbinary Problem Representation.* PhD thesis, University of London.

[82] David B. Fogel. Revisiting bremermann's genetic algorithms: I. simultaneous mutation of all parameters. *Proceeding of the 2000 Congress on Evolutionary Computation.*, 2:1204–1202, 2000.

[83] David.B. Fogel. *Evolutionary Computation, Toward a new philosophy of machine intelligence*, volume IEEE Press. 1995.

[84] L.J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[85] L.J. Fogel and G.H. Burgin. Competitive goal-seeking through evolutionary programming. Technical report, Air Force Cambridge Research Labs, 1969.

[86] C.M. Fonseca and P.J. Fleming. Multiobjective genetic algorithms. *Genetic Algorithms for Control Systems Engineering, IEE Colloquium*, 1993.

[87] B.R. Fox and M.B McMahon. Genetic operators for sequencing problems. *Gregory J.E.Rawlings(ed.).Foundations of genetic algorithms*, pages 284–300, 1991.

[88] Oliver François and Christian Lavergne. Design of eolutionary algorithms- a statistical perspective. *IEEE Transactions On Evolutionary Computation*, 5(5):129–148, April 2001.

[89] Erich Gamma, Richard Helm, Johnson Ralph, and john Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company, Inc., May 1994.

[90] Fco. Garrido and Sanz-Bobi Javier. Learning rules from the experience of an expert system using genetic algorithms. *GALESIA97. Second International Conference on Genetic Algorithms In Engineering Systems: Innovation And Applications*, (446), September 1997.

[91] Mario Girsch. *Reactive Scheduling.* PhD thesis, Vienna University of Technology, Austria., October 2001.

[92] Mario Girsh. Optimierung von schedules mit genetischen Algorithmen und iterativer Vertiefung. Master's thesis, TU WIEN, AUSTRIA, 1994.

[93] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* 1989.

[94] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithm. Technical Report 93004, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, February 1993.

[95] Marin Golub. An implementation of binary and floating chromosome representation in genetic algorithm. *Proc. Of the 41 st Annual Conference KOREMA, Opatija*, pages 93–96, 1996.

[96] Marin Golub and Domagoj Jakobovic. Adaptove genetic algorithm. *Proceedings of the 20 th International Conference ITI 98. Pula*, pages 519–524, 1998.

[97] Belzyt Gonzalez, Michel Torres, and Jose A. Moreno. A hybrid genetic algorithm approach for the "no-wait" flowshop scheduling problem. *GALESIA95. First International Conference on Genetic Algorithms In Engineering Systems: Innovation And Applications*, (414):59–64, September 1995.

[98] David Greenhalgh and Stephen Marshall. Convergence criteria for genetic algorithms. *SIAM Journal on Computing*, 30(1):269–282, May 2000.

[99] G.W. Greenwood. Finding solutions to np problems: Philosophical differences between quantum and evolutionary search algorithms. *Evolutionary Computation*, 2, 2001.

[100] Mahesh C. Gupta, Yash P. Gupta, and Anup Kumar. Genetic algorithms application in a machine scheduling.

[101] R. Georges Harik, G. Lobo Fernando, and E.David Goldberg. The compact genetic algorithm. *IEEE TRANSACTION ON EVOLUTIONARY COMPUTATION*, 3(4):187–297, November 1999.

[102] Emma Hart and Peter Ross. Gavel- a new tool for genetic algorithm visualization. *IEEE TRANSACTION ON EVOLUTIONARY COMPUTATION*, 5(4):335–348, August 2001.

[103] Thomas Dunlop Haynes. *Collective Adaptation : The sharing of Building Blocks.* PhD thesis, The University of Tulsa, 1998.

[104] Robert B. Heckendorn, Soraya Rana, and Darrell Whitley. Nonlinearity, hyperplane ranking and the simple genetic algorithm. *In Foundations of Genetic Algorithms 4*, April 1997.

[105] Robert B. Heckendorn, Soraya Rana, and Darrell Whitley. Polynomial time summary statistics for a generalization of maxsat. *Proceedings of the Genetic and Evolutionary Computation conference(GECCO-99)*, pages 281–288, 1999.

[106] Robert B. Heckendorn, Soraya Rana, and Darrell Whitley. Test function generators as embeded landscapes, foundations of genetic algorithms 5. *Collin Reeves and Wolfgang Banzhof ed., Morgan Kaufmann*, 1999.

[107] Shinn-Ying Ho, Li-Sun Shu, and Hung-Ming Chen. Intelligent genetic algorithm with a new intelligent crossover using orthogonal arrays. *Proceedings on Genetic and Evolutionary Computation Conference*, pages 289–296, 1999.

[108] J. Holland. Adaptation in natural and artificial systems. *Ann Arbor: University of Michigan Press*, 1975.

[109] Kim G. Hwan and Lee C.S. George. Genetic reinforcement learning approach to the heterogeneous machine scheduling problem. *Transactions on Robotics and Automation*, 14(6), December 1998.

[110] W.H. Ip, Y. Li, K.F. Man, and K.S. Tang. Multi-product planning and scheduling using genetic algorithm approach. *Computers and Indusitrial Engineering*, 38:283–296, 2000.

[111] Mikkel T. Jensen. Omproving robusness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing*, pages 1–18, 2001.

[112] S.K. Tamiz M. Jones, D. F. Mirrazavi. Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European Journal of Operational Research*, 137:1–9, February 2002.

[113] T. Kalinowski. Solving the task scheduling problem using a parallel genetic algorithm implemented with grade. *Computers and Artificial Intelligence*, 17:495–506, 1998.

[114] Roger M. Kerr. Scheduling in the steel industry. Technical report, Christian Doppler Laboratory for Expert Systems. CD-S 94/16, Vienna University of Technology, Austria, 1994.

[115] Hyunchul Kim, Yasuhiro Hayashi, and Koichi Nara. An algorithm for thermal unit maintenance scheduling through combined use of ga sa and ts. *Transactions on Power Systems*, 12(1), February 1997.

[116] Michael Kirley. Mea: A metapopulation evolutionary algorithm for multi-objective optimisation problems. *PROCEEDING ON EVOLUTIONARY COMPUTATION*, 2:949–952, 2001.

[117] Dimitri Knjazew. Application of the fast messy genetic algorithm to permutation and scheduling problems. Master's thesis, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, May 2000.

[118] Dimitri Knjazew and David E. Goldberg. Omega - ordering messy ga: Solving permutation problems with the fast messy genetic algorithm and random keys. Technical report, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, January 2000.

[119] R. Knosala and T. Wal. A production scheduling problem using genetic algorithm. *Journal of Materials Processing Technology*, 109:90–95, February 2001.

[120] R.John Koza. *Genetic Programming: On the Programming of Computer by means of Natural Selection.* 1992.

[121] Ibrahim Kuscu. Promotion generalization of learned behaviours in genetic programming. *Parallel Problem Solving from Nature-PPSN V. 5th International conference Amsterdam, The Netherlands*, pages 491–500, September.

[122] S.S. LAM, K.W.C. TANG, and X CAI. Genetic algorithm with pigeon-hole coding schema for sequencing problems. *Applied Artificial Intelligence*, 10:239–256, 1996.

[123] W.B. Langdon. Scheduling maintenance of electrical power transmission network using genetic programming. *Genetic programming conference*, 1996.

[124] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for travelling salesman problems. a review of representations and operators. *Artificial Intelligence Review 13*, pages 129–170, 1999.

[125] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. On the convergence and diversity-preservation properties of multi-objective evolutionary algorithms. Technical Report 108, Computer Engineering and Networks Laboratory, Swiss Federal Institut Of Technology (ETH) Zurich, Switzerland, June 2001.

[126] Albert Laura A. and David E. Goldberg. Efficient evaluation genetic algorithms under integrated fitness functions. Technical Report 2001024, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, July 2001.

[127] G. Lawton. Genetic algrotihms for schedule optimization. *AI Expert*, May 1992.

[128] In Lee. Artificial intelligence search methods for multi-machine two-stage scheduling with due date penalty, inventory, and maching costs. *Computers and Operations Research*, 28:835–852, August 2001.

[129] Richard C. Lee and Willian M. Tepfenhart. *UML and C++. A practicle Guide To Object-Oriented Development*. Printice-Hall, 1997.

[130] Suk Lee, Sang Ho Lee, Kyung Chang Lee, Man Hyung Lee, and Fumio Harashima. Intelligent performance management of of networks for advanced manufacturing systems. *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, 48(4):731–741, AUGUST 2001.

[131] Sang Lee Ho, Man Lee Hyung, and Fumio Harashima. Intelligent performance management of networks for advanced manufacturing systems. *IEEE TRANSACTION INDUSTRIAL ELECTRONICS*, 48(4):731–741, August 2001.

[132] Alexandre Leonhardi, Wolfgang Reissenberger, Tim Schmelmer, Karsten Weicker, and Nicole Weicker. Development of problem-specific evolutionary algorithms. *Parallel Problem Solving from Nature PPSN V 5th International conference Amsterdam The Netherlands*, pages 389–397, September 1998.

[133] Hitoshi lima and Nobuo Sannomiya. Module type genetic algorithm for modified scheduling problems with worker allocation. *Proceedings of the American Control Conference Arlington, VA*, pages 856–861, June 2001.

[134] Man Lin, Lars Karlsson, and Laurence Tianruo Yang. Heuristic techniques: Scheduling partially ordered tasks in a multi-processor environment with tabu search and genetic algorithms. *Seventh International Conference on Parallel and distributed Systems*, pages 515–523, 2000.

[135] Man Lin and Laurance Tianruo Yang. Hybrid genetic algorithms for scheduling partially orderedctasks in a multi-processor environment. *Sixth International Conference on Real Time Computing Systems and Applications, RTCSA99.*, pages 382–387, 1999.

[136] Bernice Sacks Lipkin. *Latex For Linux.* Springer Verlag, New York, 1999.

[137] Stanley B. Lippman. *C++ Primer.* 1989.

[138] Yutian Liu, Li Ma, and Jianjun Zhang. Ga/sa/ts hybrid algorithms for reactive power optimization. *Power Engineering Society Summer Meeting*, 1, 2000.

[139] G. Lobo Fernando and David E. Goldberg. The parameter-less genetic algorithm in practice. Technical Report 2001022, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, 2001.

[140] Xudong Luo, Ho-fung Leung, and Ho-man Lee. A multi-agent framework for meeting scheduling using fuzzy constraints. *Fourth International Conference on Multi Agent Systems*, pages 409–410, 2000.

[141] Samir W. Mahfoud. Crowding and presentation revisited. *Parallel Problem Solving From Nature, 2. R. Manner and B. Manderick (eds).*, pages 27–36, 1992.

[142] Samir W. Mahfoud. Simple analytical models of genetic algorithms for multimodal function optimization. Technical Report 93001, Department of Computer Science, University of Illinois at Urbana-Champaign. 1304 West Springfield Avenue Urbana, IL 61801, February 1993.

[143] Harpal Main, Kishan Mehrotra, Chilukuri Mohan, and Sanjay Ranka. Knowledge-based nonuniform crossover. *IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on Evolutionary Computation.*, 1:22–27, 1994.

[144] A.H. Mantawy, Yousef L. Abdel-Magid, and Shorki Z. Selim. Integrating genetic algorithms, tabu search, and simulated annealing for the unit commitment problem. *Transactions on Power Systems*, 14(3), August 1999.

[145] Wall Bartschi Matthew. Galib: A C++ Library of Genetic Algorithm Components. Technical report, Department of Mechanical Engineering. Massachusetts Institute of Technology (MIT)., Massachusetts Institute of Technology, August 1996.

[146] Wall Bartschi Matthew. *A genetic Algorithm for Resource-Constrained Scheduling.* PhD thesis, Department of Mechanical Engineering. Massaschusetts Institute of Technology (MIT)., Massaschusetts Institute of Technology, 1996.

[147] Ole J. Mengshoel and David E. Goldberg. Probabilistic crowding: Deterministic crowding with probabilistic replacement. Technical Report 99004, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, January 1999.

[148] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution programs.* Springer, 1992.

[149] Zbigniew Michalewicz. Genetic algorithms, numerical optimization, and constraints. *Ptoceedings of the 6 th International Conference on Genetic Algorithms, Pittsburgh*, pages 151–158, july 1995.

[150] Zbigniew Michalewicz. The significance of the evaluation function in evolutionary algorithms. *Proceedings of the Workshop on Evolutionary Algorithms*, pages 151–166, 1998.

[151] Zbigniew Michalewicz. Your brains and my beauty: Parent matching for constrained optimisation. *Proceedings of the 5 th International Conference on Evolutionary Computation. Anchorage, Alaska.*, pages 810–815, May 1998.

[152] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. Technical Report 95006, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, July 1995.

[153] Li Mingqiang, Kou Jisong, and Dai Lin. Ga-based multi-objective optimization. *Proceedings of the 3rd World Congress on Intelligent Control and Automation*, pages 637–640, 2000.

[154] Koji Morikawa, Tekeshi Furuhashi, and Yoshiki Uchikawa. Evolution of cim system with genetic algorithm. *IEEE World Congress on Comutational Intelligence*, 2:746–749, 1994.

[155] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolutionary algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[156] Heinz Mühlenbein. *Evolutionary Algorithms: Theory and Apllications.* New York Oxford, Oxford University Press, 1996.

[157] T. MURATA, H. ISHIBUCHI, and H. TAKANA. Genetic algorithms for flowshop scheduling problems. *Computers ind. Engng.*, 30(4):1061–1071, 1996.

[158] Nysret Musliu. *Intelligent Search Methods for Workforce Scheduling: New Ideas and Practical Applications.* PhD thesis, Vienna University of Technology, Austria., September 2001.

[159] R. Nakano. Conventional genetic algorithm for job shop problems. *Fourth International Conference on Genetic Algorithm, San Diago*, 1991.

[160] André Neubauer. The circular schema theorem for genetic algorithms and two-proint crossover. *Genetic Algorithms in Engineering Systems: Innovations and Applications*, 2-4(446):209–214, 1997.

[161] Michael O. Odetayo. Optimization population size for genetic algorithms: An investigation. *Genetic Algorithms for Control Systems Engineering, IEE Colloquium*, 1993.

[162] Christopher K. Oei, David E. Goldberg, and Shau-Jin Chang. Tournament selection niching, and the preservation of diversity. Technical Report 91011, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, 1991.

[163] S. Oussedik, D. Delahaya, and M. Schönauer. Dynamic air traffic planning by genetic algorithms. 1999.

[164] C.H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[165] C.Hicks Pongcharoen, P.M. Braiden, and D.J. Stewardson. Determining optimum genetic algorithm parameters for scheduling the manufacturing and assembly of complex product. *International Journal of Production Economics*, March 2002.

[166] Terry Quatrani. *Visual Modeling with Rational Rose and UML.* Addison Wesley Longman, 1998.

[167] Soraya Rana, Robert B. Heckendorn, and Darrell Whitley. A tractable walsh analysis of sat and its implications for genetic algorithms. *Proceedings of the fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 392–397, 1998.

[168] Magnus Rattray and Jonathan L Shapiro. The dynamic of genetic algorithm for a simple learning problem. *J.Phys. A:Math. Gen*, 29:7451–7473, 1996.

[169] I. Rechenberg. Evolution strategies: Optimierung technischer systeme nach prinzipien der biologische evolution. 1973.

[170] Colin R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* McGraw-Hill Book Company Europe, Shoppenhangers Road. Maidenhead. Berkshire SL6 2QL, 1995.

[171] A.P. Reynolds, V.J. Rayward-Smith, and G.P. McKeown. The application of simulated annealing to an industrial scheduling problem. *GALESIA97. Second International Conference on Genetic Algorithms In Engineering Systems: Innovation And Applications*, (446):345–350, September 1997.

[172] A. Roach and R. NAGI. A hybrid ga-sa algorithm for just-in-time scheduling of multi-level assemblies. *Computers ind. Engng.*, 30(4):1047–1060, 1996.

[173] Franz Rothlauf, David E. Goldberg, and Armin Heinzl. Bad codings and the utility of well-designed genetic algorithm. Technical Report 2000007, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, January 2000.

[174] Alvaro Ruiz-Andino, Lourdes Araujo, Fernando Sacuteaenz, and José Ruz. A hybrid evolutionary approach for solving constrained optimization problems over finite domains. *IEEE Transactions On Evolutionary Computation*, 4(4):353–372, November 2000.

[175] J. Rumbaugh. *Object Oriented modelling and design.* Englewood Cliffs, NJ., 1991.

[176] Johannes Sametinger. *Software Engineering With Reusable Components.* Berlin[u.a.], 1997.

[177] H. Sayoud, K. Takahashi, and B. Vaillant. Designing communication networks topologies using steady-state genetic algorithms. *IEEE COMMUNICATION LETTERS*, 5(3):113–115, MARCH 2001.

[178] Florian Schmitt and Franz Rothlauf. On the importance of the second largest eigenvalue on the convergence rate of genetic algorithms. 2001021, Illinois Genetic Algorithm Laboratory. Department of General Engineering, University of Illinois at Urbana-Champaign. 117 Transportation Building. 104 South Mathews Avenus. Urbana, IL 61801, June 2001.

[179] H.P. Schwefel. *Evolution als strategie der Experimentellen Forschung in der Strömungstechnik.* PhD thesis, Technical University of Berlin, 1965,.

[180] K.J. Shaw and P.J. Fleming. Including real-life problem preferences in genetic algorithms to improve optimisation of production schedules. *Genetic Algorithms In Engineering Systems: Innovation And Applications*, (446):239–244, September 1997.

[181] wolfgang Slany. *Fuzzy Scheduling.* PhD thesis, Vienna University of Technology, Austria, Vienna, Austria, 1994.

[182] William M. Spears. *The Role of Mutation and Recombination in Evolutionary Algorithms.* PhD thesis, George Mason University, Fair fax, Virginia, 1998.

[183] Chris Stephens and Henri Waelbrpeck. Schema evolution and building blocks. *Conference Proccedings on Evolutionary Computation. CEC99*, 7(2):109–124, 1999.

[184] Bjarne Stroustrup. *The C++ programming language.* Reading, Massaschusetts, 1997.

[185] Patrick David Surry. *A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms.* PhD thesis, University of Edinburgh, 1998.

[186] G. Syswerda. Schedule optimization using genetic algorithms. in davis, l.(ed.). *Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold*, pages 332–349, 1991.

[187] G. Syswerda. A study of reproduction in generational and steady-state genetic algorithms. *In Foundations of Genetic Algorithms. Morgan Kauffmann Publishers.*, pages 94–101, 1991.

[188] K.C. Tan, Tong H. Lee, D. Khoo, and E.F. Khor. A multiobjective evolutionary algorithm toolbox for computer-aided multiobjective optimization. *Transactions On Systems, Man, and Cybernetics-PART B: Cybernetics.*, 31(4):537–556, August 2001.

[189] Guo Tao and Zbigniew Michalewicz. Inver-over operator for tsp. *Proceedings of the 5 th Parallel Problem Solving from Nature*, pages 803–812, September 1998.

[190] Masaru Tezuka, Masahiro Hiji, Kazunori Miyabayashi, and Keigo Okumura. A new cluster representation and common cluster crossover for job shop scheduling problems. *Proceedings on Real-World Applications of Evolutionary Computing, EvoWorkshops 2000:*, 1803:297–306, 2000.

[191] Chuan-Kang Ting, Sheng-Tun Li, and Chungnan Lee. Tga: A new integrated approach to evolutionary algorithms. *Proceedings of the 2001 Congress on: Evoulutionary Computation*, 2:917–924, 2001.

[192] Jean-Philippe Vacher, Thierry Galinho, Franck Lesage, and Alain Cardon. Genetic algorithms in a multi-agent system. *IEEE International Joint Symposia on Intelligence and Systems*, pages 17–26, 1998.

[193] Manuel Vacuteazquez and Darell Whitley. A comparison of genetic algorithms for the dynamic job shop scheduling problem. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2000.

[194] Manuel Vacuteazquez and Darell Whitley. A comparison of genetic algorithms for the static job shop scheduling problem. *Parallel Problem Solving From Nature- PPSN VI, 6 th International Conference Paris, Proceedings.*, September 2000.

[195] Dingwei Wang, K.L. Yung, and W.H. Ip. A heuristic genetic algorithm for subcontractor selection in a global manufacturinf environment. *IEEE TRANSACTION ON SYSTEMS, MAN, AND CYBERNETICS-PARTC: APPLICATION AND REVIEWS*, 31(2):189–198, MAY 2001.

[196] Kefeng Wang, Thomas Löhl, Mario Stobbe, and Sebastian Engell. A genetic algorithm for online-scheduling of multiproduct polymer batch plant. *Computers and Chemical Engineering*, 24:393–400, 2000.

[197] Michiko Watanabe, Masashi Furukawa, Akihiro Mizoe, and Tatsuo Watanabe. Ga applications to physical distribution scheduling problem. *IEEE TRANSACTION ON INDUSTRIAL ELECTRONICS*, 4(724-730), 48.

[198] J.P. Watson, L. Barbulescu, A.E. Howe, and L.D. Whitley. Algorithm performance and problem structure for flow-shop scheduling. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 1999.

[199] Darrell Whitley. A free lunch proof for gray versus binary encodings. *Proceedings of the Genetic and Evolutionary Computation Conference. GECCO-99*, 1999.

[200] Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43:817–831, December 2001.

[201] Darrell Whitley, A. E. Howe, S. Rana, J.P. Watson, and L. Barbulescu. Comparing heuristics search methods and genetic algorithms for warehouse scheduling. *In Systems, Man and Cybernetics*, 1998.

[202] Darrell Whitley, K. Mathias, and L. Pyeatt. Hyperplane ranking in simple genetic algorithms. *International Conference on Genetic Algorithms. L. Eshelman, ed. Morgan Kaufmann*, 1995.

[203] Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. *In Schaffer, J.(ed.) Proceedings on the third International Conference on Genetic Algorithms. Los Altos, CA*, pages 133–140, 1989.

[204] Darrell Whitley and N. Yoo. Modeling simple genetic algorithms for permutation problems. *Foundations of Genetic Algorithms-3 D. Whitley and M. Vose, eds. Morgan Kaufmann*, 1995.

[205] Pierre A.I. Wijkman. Solving the tsp problem with a new model in evolutionary computaion. *Genetic Algorithms on Engineering Systems: Innovations and Apllications*, (446):145–150, 1997.

[206] David H. Wolpert and Walliam G. Macready. No free lunch theorems for optimization. *IEEE Transactions On Evolutionary Computation*, 1(1):67–82, April 1997.

[207] Takeshi Yamada and Ryohei Nakano. A genetic algorithm with multi-step crossover for hob-shop scheduling problems. *GALESIA95. First International Conference on Genetic Algorithms In Engineering Systems: Innovation And Applications*, (414):146–151, September 1995.

[208] Habib Youssef, Sadiq M.Sait, and Hakim Adiche. Evolutionary algorithms, simulated annealing and tabu search: a comparative study. *Engineering Applications of Artificial Intelligence*, 14:167–181, April 2001.

[209] Tina Yu and Peter Bentley. Methods to evolve legal phenotypes. *Parallel Problem Solving from Nature-PPSN V. 5th International conference Amsterdam, The Netherlands*, pages 280–291, September 1998.

[210] F. Zhang, Y.F. Zhang, and A.Y.C. Nee. Using genetic algorithm in process planning for job shop machining. *IEEE Transactions On Evolutionary Computation*, 1(4):278–289, November 1997.

[211] J. Zhao, B. Knight, E. Nissan, and A. Soper. Fuelgen: a genetic algorithm-based system for fuel loading pattern design in nuclear power reactors. *Expert Systems with Applications 14*, pages 461–470, 1998.

[212] Hong Zhou, Yuncheng Feng, and Limin Han. The hybrid heuristic genetic algorithm for job shop scheduling. *Computers and Industrial Engineering*, 40:191–200, July 2001.

[213] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Application.* PhD thesis, Swiss Federal Institute of Technology Zurich, Switzerland, November 1999.

[214] Eckart Zitzler, Deb Kalyanmoy, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, April 2001.

[215] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2:improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory, Swiss Federal Institut Of Technology (ETH) Zurich, Switzerland, May 2001.

[216] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

CurriculumVitae

Personal Data:

Name : Dipl.-Ing. Maamar SAIB

Nationality : Algeria

Adresse : Linzer Strasse 429/4012

A-1140 Vienna

AUSTRIA

Email : saib@dbai.tuwien.ac.at

Privat Email: amarsab67@hotmail.com

Highlights of Qualifications:

- Programming Experience: C++, Turbo Pascal, Clipper, DBASE
- Object Oriented Analysis and Design: Unified Modeling Language (UML)
- Rational Rose Tool for Object Oriented Analysis and Design
- Design Patterns

Languages:

- Native Languages:
  Arabic, French.
- Other Languages:

  Written Spoken

  -German: advanced advanced

-English intermediate intermediate

Education:

- Secondary School: Lycee de Chlef, Algeria. graduated in 1986
- University: University of Science and Technology, Alger, Algeria. Major Computer Science, graduated in 1992.
- German Course: German Course, from 1996-1997 at the University of Vienna, Austria. Advance Level.
- Phd Student: Vienna University of Technology, from 1998 to 2002.

Employment:

- From June 1992 To February 1993 : Software Company In Algeria
  Project: Development of Software with Database Dbase
- from October 1992 to February 1993: Technical University In Algeria: Assistant Teaching Algorithms and Programming in Computer Science Institute.
- From February 1993 to 1996: Import Export Company(OAIC), Algeria:
  Project: Development Of Software for Production With Clipper and Dbase
- From July 2000 to September 2000: Siemens Austria, Wien.
  Project: Development of Graphical User Interface with Microsoft Visual C++ for Network management System
- From April 2001: Vienna University of Technology, Department of Pattern Recognition. Research Assistant
  Project: Geograph. Development Of New Tool with C++ for Pattern Recognition and Image processing.

Accomplishments:

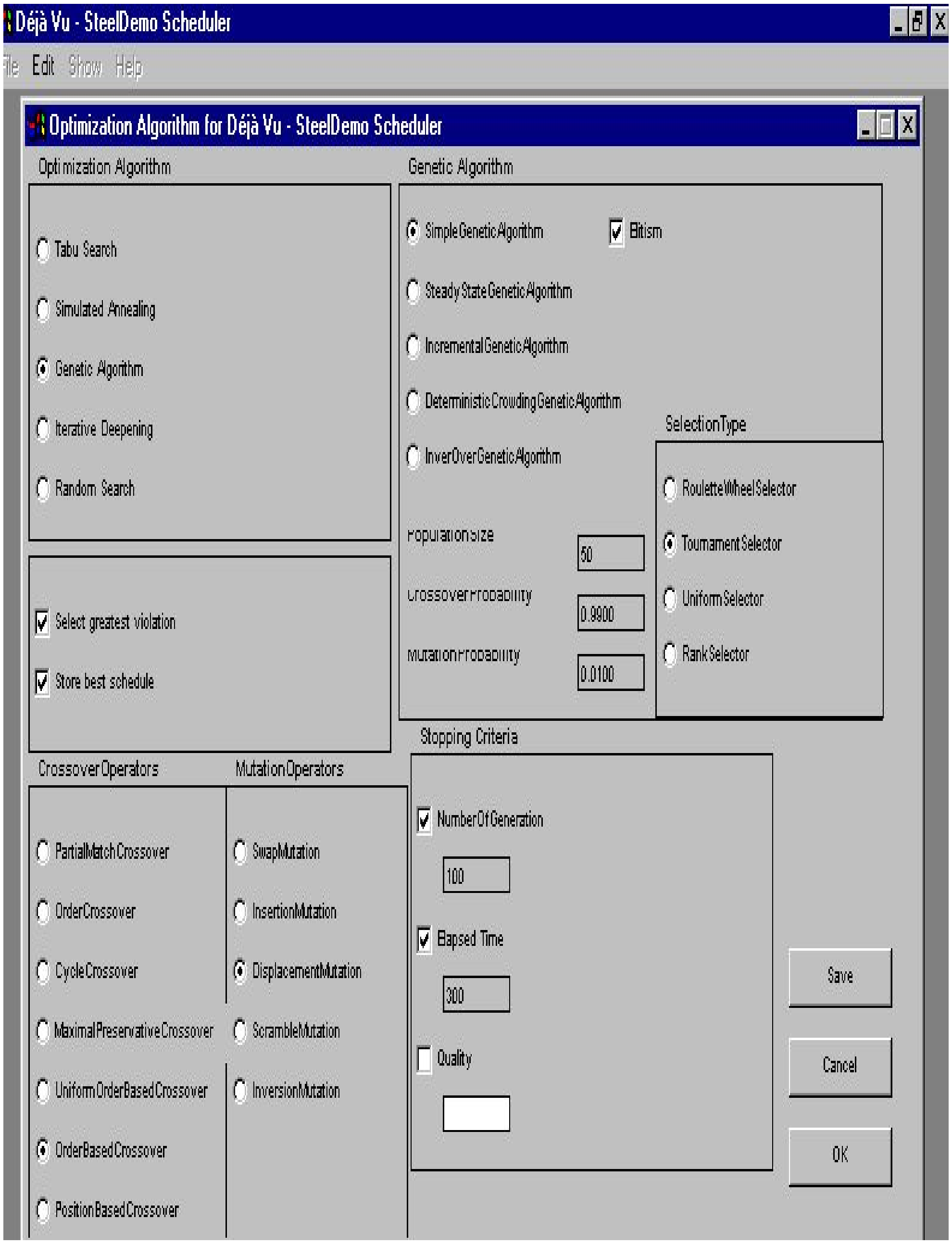
- Implementation of software for classification and Multi Criteria analysis with Turbo Pascal

- Implementation of production software with Clipper and DBASE
- Design(With UML and Rational Rose Tool) and implementation (With Microsoft Visual C++)of genetic algorithms in scheduling problems
- Design and implementation of software for Pattern Recognition and Image processing With C++ under Unix.

Publications:

- Haxhimusa Yll and Saib Maamar and Glantz Roland and Kropatsch Walter G. Equivalent Contraction Kernels Using Dynamic Trees, Proceedings of the 6th. Computer Vision Winter Workshop. Bled, Slovenia. 267-275, 2001.
- Haxhimusa Yll and Glantz Roland and Saib Maamar and Langs Georg, and Kropatsch Walter G., Reduction Factors of Image Pyramid on Udirected and Directed Graph, Proceedings of the 7th. Computer Vision Winter Workshop. Austria,2002.
- Saib Maamar and Haxhimusa Yll and Glantz Roland, *Dgc_tool: Building Irregular Graph Pyramid Using Dual Graph Contraction, PRIP, Vienna University of Technology. Tech.Rep.No.69, 2002.*

Appendix


Déjà Vu - SteelDemo Scheduler
File Edit Show Help
Optimization Algorithm for Déjà Vu - SteelDemo Scheduler
Save
Cancel
OK

Optimization Algorithm

○ Tabu Search
○ Simulated Annealing
● Genetic Algorithm
○ Iterative Deepening
○ Random Search

☑ Select greatest violation
☑ Store best schedule

Genetic Algorithm

● SimpleGeneticAlgorithm ☑ Elitism
○ SteadyStateGeneticAlgorithm
○ IncrementalGeneticAlgorithm
○ DeterministicCrowdingGeneticAlgorithm
○ InverOverGeneticAlgorithm

PopulationSize: 50
CrossoverProbability: 0.9900
MutationProbability: 0.0100

SelectionType

○ RouletteWheelSelector
● TournamentSelector
○ UniformSelector
○ RankSelector

CrossoverOperators

○ PartialMatchCrossover
○ OrderCrossover
○ CycleCrossover
○ MaximalPreservativeCrossover
○ UniformOrderBasedCrossover
● OrderBasedCrossover
○ PositionBasedCrossover

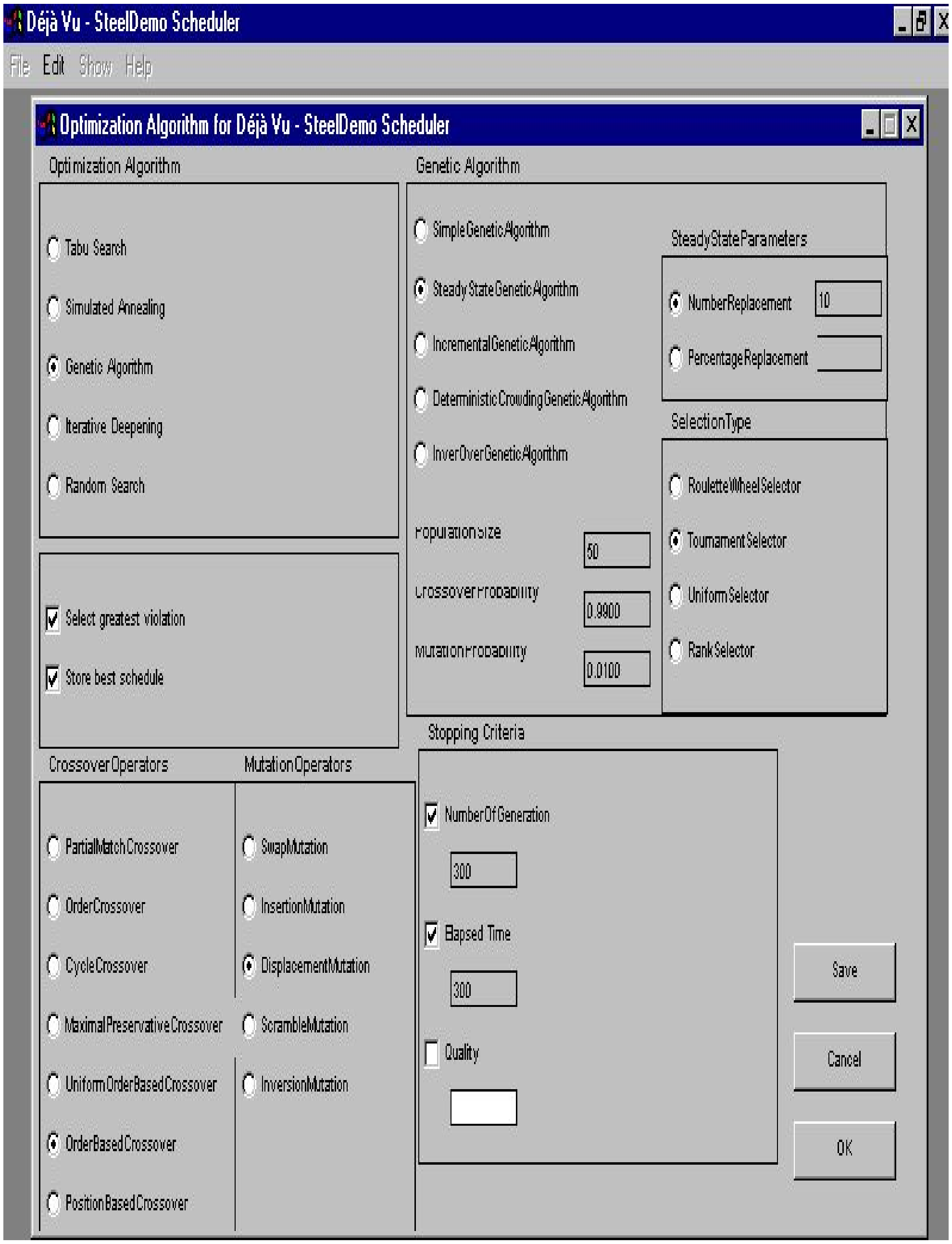
MutationOperators

○ SwapMutation
○ InsertionMutation
● DisplacementMutation
○ ScrambleMutation
○ InversionMutation

Stopping Criteria

☑ NumberOfGeneration: 100
☑ Elapsed Time: 300
□ Quality:

Figure 7.1: Graphical User Interface: Simple genetic algorithm


Déjà Vu - SteelDemo Scheduler
File
Edit
Show
Help
Optimization Algorithm for Déjà Vu - SteelDemo Scheduler
Save
Cancel
OK

Optimization Algorithm

○ Tabu Search
○ Simulated Annealing
● Genetic Algorithm
○ Iterative Deepening
○ Random Search

☑ Select greatest violation
☑ Store best schedule

Genetic Algorithm

○ SimpleGeneticAlgorithm
● SteadyStateGeneticAlgorithm
○ IncrementalGeneticAlgorithm
○ DeterministicCrowdingGeneticAlgorithm
○ InverOverGeneticAlgorithm

PopulationSize 50
CrossoverProbability 0.9800
MutationProbability 0.0100

SteadyStateParameters

● NumberReplacement 10
○ PercentageReplacement

SelectionType

○ RouletteWheelSelector
● TournamentSelector
○ UniformSelector
○ RankSelector

CrossoverOperators

○ PartialMatchCrossover
○ OrderCrossover
○ CycleCrossover
○ MaximalPreservativeCrossover
○ UniformOrderBasedCrossover
● OrderBasedCrossover
○ PositionBasedCrossover

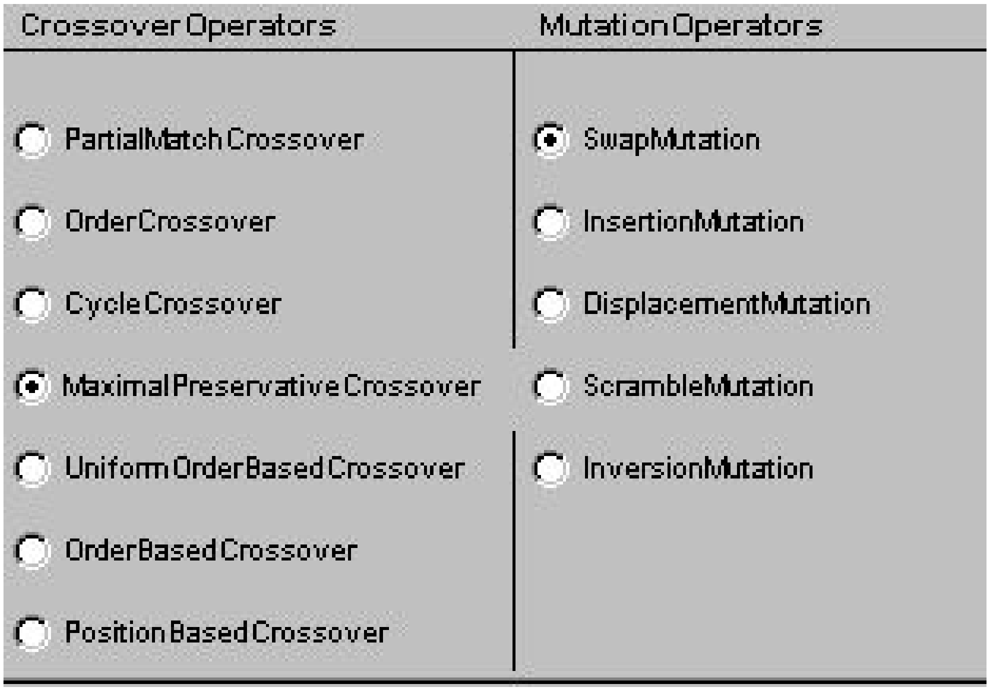
MutationOperators

○ SwapMutation
○ InsertionMutation
● DisplacementMutation
○ ScrambleMutation
○ InversionMutation

Stopping Criteria

☑ NumberOfGeneration 300
☑ Elapsed Time 300
☐ Quality

Figure 7.2: Graphical User Interface: Steady state genetic algorithm



Crossover Operators

○ PartialMatch Crossover

○ Order Crossover

○ Cycle Crossover

● Maximal Preservative Crossover

○ Uniform Order Based Crossover

○ Order Based Crossover

○ Position Based Crossover

Mutation Operators

● SwapMutation

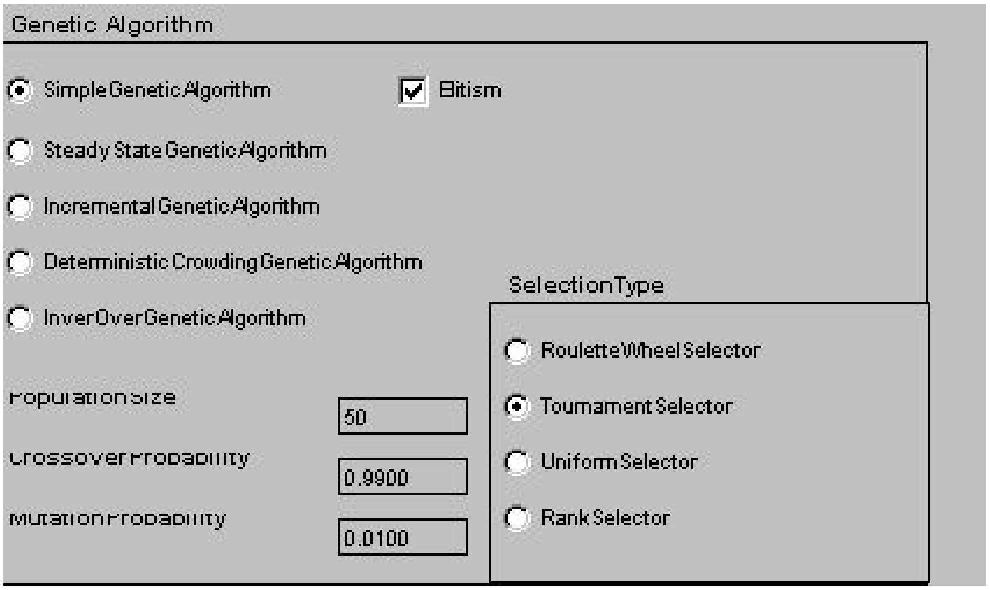
○ InsertionMutation

○ DisplacementMutation

○ ScrambleMutation

○ InversionMutation

Figure 7.3: Graphical User Interface:genetic algorithm operators



Genetic Algorithm

● Simple Genetic Algorithm

☑ Elitism

○ Steady State Genetic Algorithm

○ Incremental Genetic Algorithm

○ Deterministic Crowding Genetic Algorithm

○ InverOver Genetic Algorithm

Population Size: 50

Crossover Probability: 0.9900

Mutation Probability: 0.0100

Selection Type

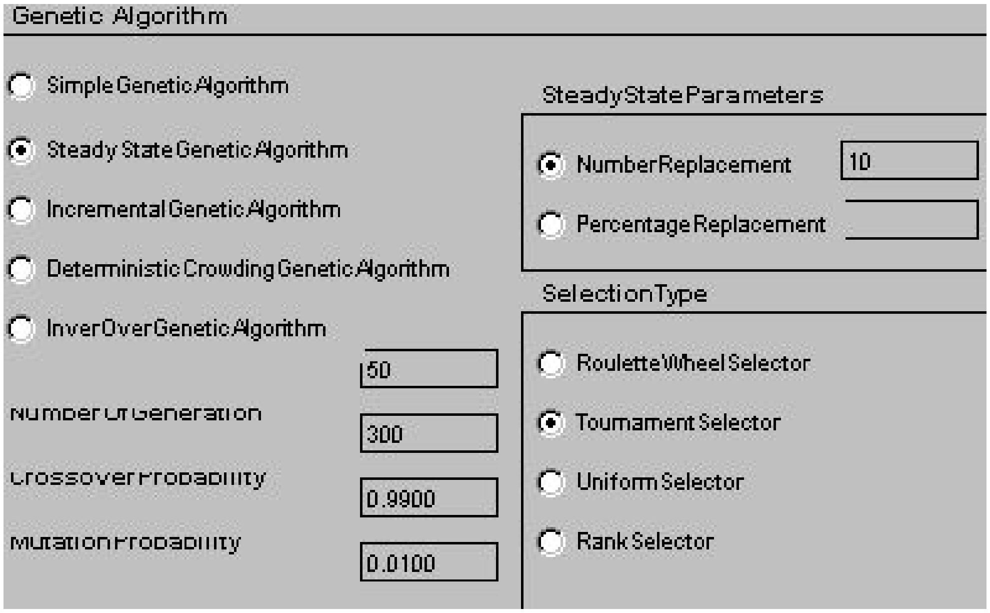
○ Roulette Wheel Selector

● Tournament Selector

○ Uniform Selector

○ Rank Selector

Figure 7.4: Graphical User Interface: Simple genetic algorithm parameters



Genetic Algorithm

○ Simple Genetic Algorithm

● Steady State Genetic Algorithm

○ Incremental Genetic Algorithm

○ Deterministic Crowding Genetic Algorithm

○ InverOver Genetic Algorithm

50

Number Of Generation 300

Crossover Probability 0.9900

Mutation Probability 0.0100

Steady State Parameters

● Number Replacement 10

○ Percentage Replacement

Selection Type

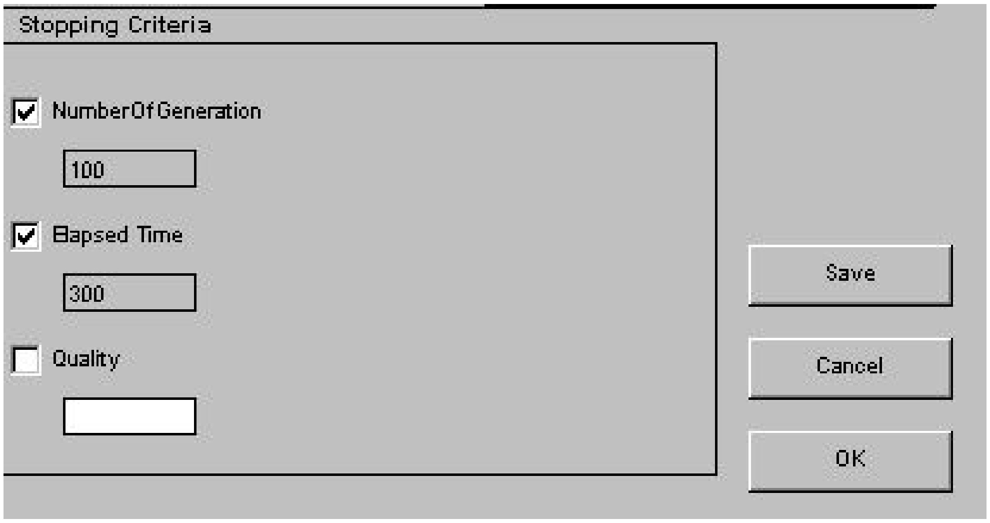
○ Roulette Wheel Selector

● Tournament Selector

○ Uniform Selector

○ Rank Selector

Figure 7.5: Graphical User Interface: Steady state genetic algorithm parameters



Stopping Criteria

☑ NumberOfGeneration

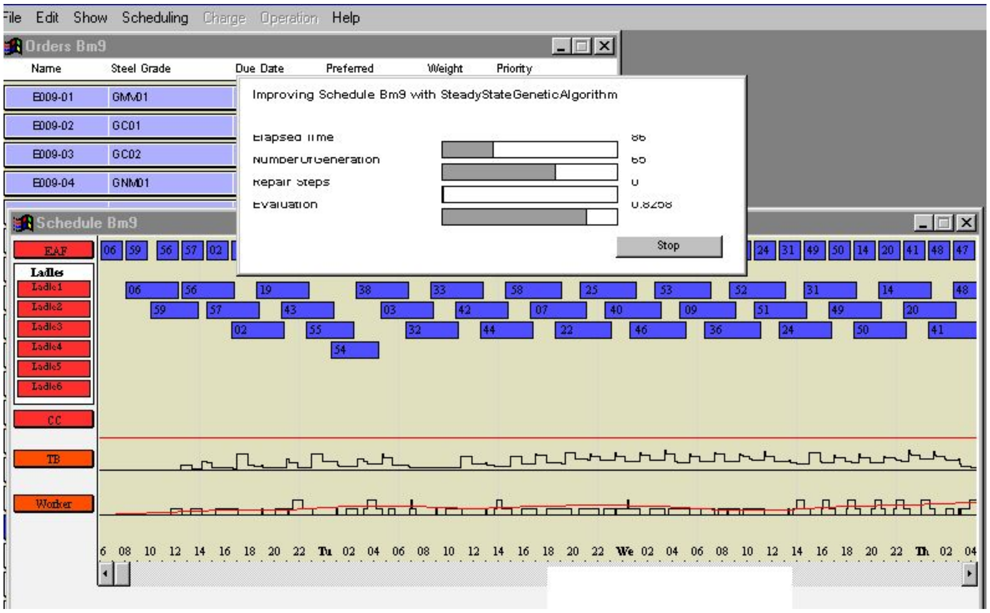
100

☑ Elapsed Time

300

☐ Quality

Figure 7.6: Graphical User Interface: Stoping Criteria

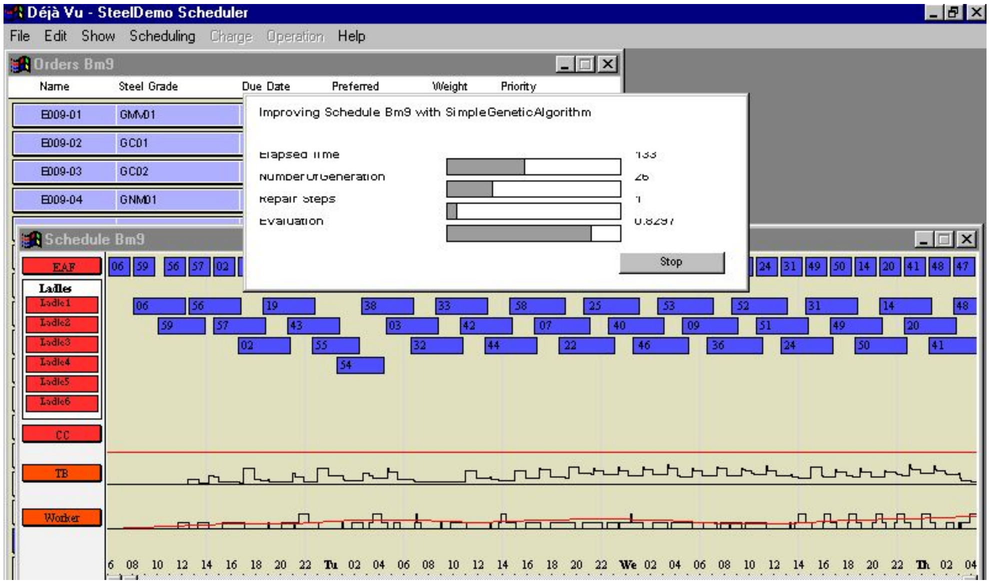


Name	Steel Grade	Due Date	Preferred	Weight	Priority
E009-01	GMV01				
E009-02	GC01				
E009-03	GC02				
E009-04	GNM01				

Improving Schedule Bm9 with SteadyStateGeneticAlgorithm

Elapsed Time 86
NumberOfGeneration 65
Repair Steps 0
Evaluation 0.8258

Figure 7.7: Graphical User Interface: Improvement with Steady State GA



Name	Steel Grade	Due Date	Preferred	Weight	Priority
E009-01	GMV01				
E009-02	GC01				
E009-03	GC02				
E009-04	GNM01				

Improving Schedule Bm9 with SimpleGeneticAlgorithm

Elapsed Time 133
NumberOfGeneration 26
Repair Steps 1
Evaluation 0.8297

Figure 7.8: Graphical User Interface: Improvement with Simple GA