Babeş-Bolyai University

Cristina Mihăilă

PhD Thesis

Evolutionary Computation in Scheduling

Supervisor
D. Dumitrescu

Cluj-Napoca, 2011

Abstract

The concept of “scheduling” is not new: Sun Tzu wrote about scheduling and strategy 5000 years ago from a military perspective, the pyramids are over 3000 years old, transcontinental railways have been being built for some 200 years etc – none of these activities could have been accomplished without some form of scheduling, i.e. the understanding of activities and sequencing [Weaver2006].

Today, scheduling is a form of decision making that plays an important role in many fields: from personal (designing an itinerary of personal development or setting up a daily agenda) to governmental decision making (setting up a strategy for economic growth or a strategy for assuring the quality of education), from cultural (organizing an exhibition or producing a movie) to economic decision making (introducing a new product on the market or relocating a factory).

Scheduling is now studied by researchers in management, industrial engineering, operations research and computer science. Given the importance, and the complexity, of the scheduling problems this thesis tries to investigate if the paradigm of evolutionary computation, in essence a heuristic one, is a good candidate to better and/or faster solve some instance of various scheduling problems.

Keywords: scheduling, evolutionary algorithm

List of publications

Dumitrescu, D., Iantovics, B., **Florea, C.**, Multi-Agent Systems: a new allocation protocol and evolutionary search for equilibrium; , in *Proceedings of Symposium "Zilele academice clujene" - Computer Science Section*, 119-133, Cluj-Napoca, Romania, 2002.

Dumitrescu, D., **Florea, C.**, Patranjan; P., Evolutionary Reorganization in MAS; , in *Proceedings of the European Conference on Information Technology* (ECIT02), 1-5, Iasi, Romania, 2002.

Dumitrescu, D., **Florea, C.**, Patranjan; P., A New Evolutionary Model for Multi Agent Systems, in *Proceedings of* the *4th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC02), 137-143, Timisoara, Romania, 2002.

Groşan, C., Oltean, M., **Florea, C.**, NP-complete problems using Evolutionary Algorithms, *Lucrarile Seminarului de Didactica Matematicii al Universitatii Babes-Bolyai*, Vadu-Crisului, Romania, 2003.

Florea, C., Dumitrescu, D., Negotiation in Multiagent Systems; in *Proceedings of Conference on Applied and Industrial Mathematics* (CAIM03), Oradea, Romania, 2003.

Mihaila, C., Dumitrescu, D., Quantum Computing and Multiagent Systems, in *Proceedings of the Symposium "Colocviul Academic Clujean de Informatica",* Cluj-Napoca, Romania, 2005.

Mihiş, A., Creţu, C., **Mihăilă, C.**, Şerban, C., Code Simplification using Boolean Functions Simplification, in *Proceedings of the International Conference of Mathematics & Informatics*, Supplement of "Studii □i cercetări □tiin□ifice. Seria Matematică", no.16, University of Bacau, 493-502, Bacău, România, 2006.

Niţchi, I.Ş., Mihăilă, A., **Mihăilă, C.**, About Project Management Planning Optimization using Genetic Algorithms, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 79-82 ,Cluj-Napoca, România, 2007.

Niţchi, I.Ş., Avram-Niţchi, R., Mihăilă, A., **Mihăilă, C.**, About the Logical Model for Intelligent Agents, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 83-90, Cluj-Napoca, România, 2007.

Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A., **Mihăilă, C.**, On the collaborative systems for e-business, in *Proceedings of the International Conference on Competitiveness and European Integration*, 266-272, Cluj-Napoca, România, 2007.

Mihăilă C., Cobârzan C., Evolutionary approach for multimedia caching, in *IEEE Proceedings of the Evolutionary Techniques in Data Processing Workshop,* International Conference on Database and Expert Systems Application (DEXA), 531-536, Torino, Italia, 2008

Mihăilă C., Mihăilă A., An Evolutionary Algorithm for Uniform Parallel Machines Scheduling, in *IEEE Proceedings of the European Modelling Symposium*, 76-80, Liverpool, United Kingdom, 2008.

Cobârzan C., **Mihăilă C.**, A Genetic Algorithm for Utility Based Video Proxy-Caching, in *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 231-238, Timişoara, România, 2008

Mihăilă A., Mihiş A., **Mihăilă C.,** Genetic Algorithm for Logical Topic Text Segmentation, in *IEEE Proceedings of the International Conference on Digital Information Management,* 500-505, London, United Kingdom, 2008

Mihăilă A., **Mihăilă C.**, Uniform Parallel Machines Scheduling using an Evolutionary Algorithm, in *IEEE Proceedings of the International Workshop on Evolutionary Multiobjective Optimization Design and Applications*, International Conference on Intelligent Systems Design and Applications (ISDA), 401-406, Kaohsiung, Taiwan, 2008

Mihăilă C., Nițchi, I.Ş., R., Mihăilă, A., Coroş R., A genetic algorithm for permutation flow shop scheduling problem, in Annals of Tiberiu Popoviciu Seminar of Functional Equation, Approximation and Convexity, p. 241-250, Cluj-Napoca, România, 2008

Oltean M., Groşan C., Dioşan L., **Mihăilă C.,** *Genetic Programming with Linear Representation a Survey,* International Journal on Artificial Intelligence Tools, 197-238, 2009

Table of contents

Introduction

In recent years, scheduling research has had an increasing impact on practical problems, and a range of scheduling techniques have made their way into real-world application development. Constraint-based models now couple rich representational flexibility with highly scalable constraint management and search procedures. Similarly, mathematical programming tools are now capable of addressing problems of unprecedented scale, and meta-heuristics provide robust capabilities for schedule optimization. With these mounting successes and advances, it might be tempting to conclude that the chief technical hurdles underlying the scheduling problem have been overcome. However, such a conclusion (at best) presumes a rather narrow and specialized interpretation of scheduling, and (at worst) ignores much of the process and broader context of scheduling in most practical environments [Smith2005].

Summarizing the current state of the art several technological strengths can be identified [Smith2005]:

- scalability – current scheduling techniques are capable of solving large problems (i.e., tens of thousands of activities, hundreds of resources) in reasonable time frames.
- modeling flexibility – current techniques are capable of generating schedules under broad and diverse sets of temporal and resource capacity constraints.
- optimization – research in applying various global, local and meta-heuristic based search frameworks to scheduling problems has produced a number of general approaches to schedule optimization, and increasing integration of AI-based search techniques with mathematical programming tools (e.g., linear, mixed-integer constraint solvers) is yielding quite powerful optimization capabilities.

Despite the strengths of current techniques, the problems being addressed are generally NP hard and solved only approximately; there is considerable room for improvement in techniques for accommodating different classes of constraints and for optimizing under different sets of objective criteria [Smith2005].

The aim of this thesis is to investigate the use of evolutionary computation techniques in solving different classes of scheduling problems. In this respect, the first part of the thesis (chapters 1-3) presents a synthesis of some theoretical concepts from scheduling theory and evolutionary computation while the second part (chapters 4-6) introduces some personal results obtained in applying evolutionary computation techniques to some class of scheduling problems.

Chapter 1 introduces some basic notations (tasks, resources, objective functions) and concepts used in scheduling theory and a classification scheme used for codifying scheduling problems. Some aspects related to scheduling problems like communication delay and multi-processor task, scheduling with limited processor availability, scheduling under resource constraints and multicriteria scheduling are also presented.

Chapter 2 illustrates the complexity of scheduling problems. The polynomial reduction and the complexity hierarchy are also presented.

Chapter 3 is dedicated to evolutionary computation techniques and underlies their working mechanisms. Some aspect related to the main elements (representation, fitness function, selection, crossover, mutation, parameters) that influence the performance of an evolutionary algorithm are presented.

Chapter 4 presents a genetic algorithm for solving a uniform parallel machines scheduling problem. Uniform machines are special classes of resources in which machines have different speeds but the speed is constant and does not depend on tasks.

Chapter 5 presents some results obtained by applying a hybrid genetic algorithm in order to find a solution of a permutation flow-shop scheduling problem. The objective of a permutation flow shop scheduling problem is to find a sequence for processing a set of jobs using a set of machines such that a given criterion is optimized, taking into account that each machine processes the jobs in the same order.

Chapter 6 presents a video proxy-caching scheduling problem along with some results obtained for determining the coefficients of utility function which is at the core of the cache replacement mechanism.

The main contributions of this thesis consist in:

- a new genetic algorithm for the uniform parallel machines scheduling problem [Mihăilă&Mihăilă2008a]. The proposed algorithm not only obtains better results than other algorithms, but it also computes the result faster [Mihăilă&Mihăilă2008b].
- a new hybrid genetic algorithm for permutation flow shop scheduling problem [Mihăilă et.al.2008b]. The novelty of the proposed algorithm consists in using a combination of a random initialization procedure and an initialization procedure based on NEH construction heuristic, in defining a new crossover operator and in using a mutation operator defined as a combination between an operator based on NEH construction heuristic shift mutation. The results obtained by the proposed algorithm are comparable with the results obtained by other algorithms.
- two new ways of defining the utility of the objects stored inside a video proxy-cache and a new genetic algorithm used for determining the coefficients that appear in these two definitions with the aim of maximizing the byte hit rate [Mihăilă&Cobârzan2008]. The obtained results in terms of byte hit rate when the proposed algorithm with one of the utility function were used are similar or even better than those obtained by other metric [Cobârzan&Mihăilă2008].

Part A – Background

1. Scheduling problems

The aim of this chapter is to introduce the basic elements of a scheduling problem (tasks, resources and objective functions) and some aspects related to these elements. A classification scheme of scheduling problem is also presented.

Scheduling is concerned with the allocation of scarce resources to activities with the objective of optimizing one or more performance measure [Leung2004].

1.1. Deterministic scheduling problem

Scheduling problems are characterized by three sets [Blazewicz2007]: set $T = \{T_1, T_2, \dots, T_n\}$ of n tasks, set $P = \{P_1, P_2, \dots, P_m\}$ of m processors or machines and set $R = \{R_1, R_2, \dots, R_s\}$ of s types of *additional resources*. Scheduling, generally speaking, means to assign processors from P and (possibly) resources from R to tasks from T in order to complete all tasks under the imposed constraints.

There are two general constraints in classical scheduling theory [Blazewicz2007]: each task is to be processed by at most one processor at a time (plus possibly specified amounts of additional resources) and each processor is capable of processing at most one task at a time (note that this constraint may be relaxed).

The processors may be either parallel, i.e. performing the same functions, or dedicated i.e. specialized for the execution of certain tasks [Blazewicz2007]. Three types of parallel processors are distinguished depending on their speeds [Blazewicz2007]: identical (if all processors from set P have equal task processing speeds), uniform (if the processors differ in their speeds, but the speed b_i of each processor is constant and does not depend on the task in T) and unrelated (if the speeds of the processors depend on the particular task processed). In case of dedicated processors there are three models of processing sets of tasks [Blazewicz2007]: flow shop, open shop and job shop.

In general, task $T_j \in T$ is characterized by the following data [Blazewicz2007]:

- *vector of processing times $p_j = [p_{1j}, p_{2j}, \dots, p_{mj}]^T$*, where p_{ij} is the time needed by processor P_j to process T_j.
- *arrival time* (or ready time) r_j, which is the time at which task T_j is ready for processing. If the arrival times are the same for all tasks from T, then it is assumed that $r_j = 0$ for all j.
- *due date d_j*, which specifies a time limit by which T_j should be completed; usually, penalty functions are defined in accordance with due dates.
- *deadline $\tilde{d}_j$*, which is a "hard" real time limit by which T_j must be completed.
- *weight* (priority) w_j, which expresses the relative urgency of T_j.
- *resource request* (if any).

A schedule is called *preemptive* if each task may be preempted at any time and restarted later at no cost, perhaps on another processor. If preemption of all the tasks is not allowed we will call the schedule *non-preemptive* [Blazewicz2007].

In set *T precedence constraints* among tasks may be defined. $T_i \prec T_j$ means that the processing of T_i must be completed before T_j can be started. In other words, in set T a precedence relation $\prec$ is defined.

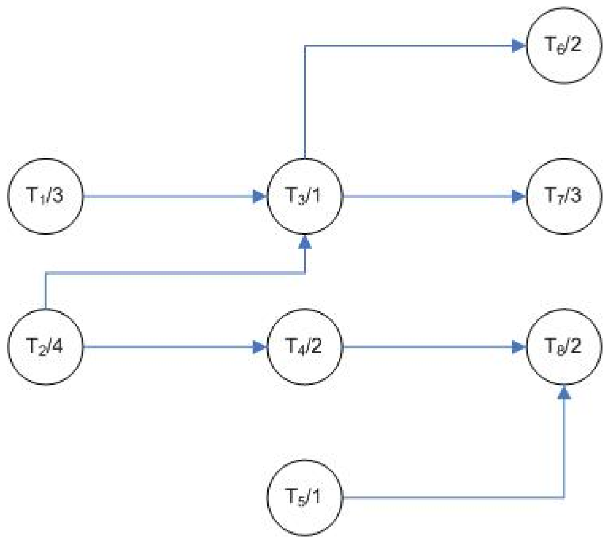


Figure 1.1 An example of task dependency [Blazewicz2007]

A *schedule* is an assignment of processors from set *P* (and possibly resources from set *R*) to tasks from set *T* in time such that the following conditions are satisfied [Blazewicz2007]:

- at every moment each processor is assigned to at most one task and each task is processed by at most one processor,
- task T_j is processed in time interval $[r_j, \infty)$,
- all tasks are completed,
- if tasks T_i, T_j are in relation $T_i \prec T_j$, the processing of T_j, is not started before T_i is completed,

- in the case of non-preemptive scheduling no task is preempted (then the schedule is called *non-preemptive*), otherwise the number of preemptions of each task is finite (then the schedule is called *preemptive*),
- resource constraints, if any, are satisfied.

Schedules may be represented by Gantt charts as shown in Figure 1.2.

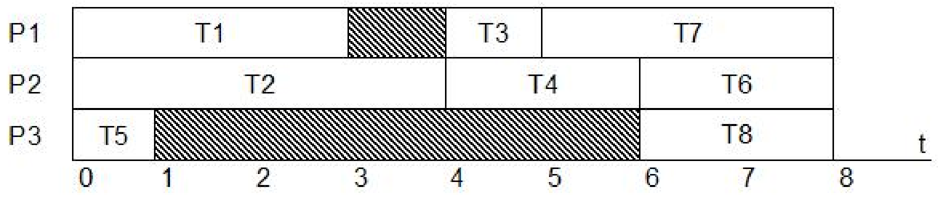


Figure 1.2 An example of Gantt chart [Blazewicz2007]

The following parameters can be calculated for each task T_j, $j = 1, 2, \ldots, n$, processed in a given schedule [Blazewicz2007]:

- *completion time* C_j,
- *flow time* $F_j = C_j - r_j$, being the sum of waiting and processing times;
- *lateness* $L_j = C_j - d_j$,;
- *tardiness* $D_j = max\ \{C_j - d_j, 0\}$;
- *earliness* $E_j = max\ \{d_j - C_j, 0\}$.

Completion-time of job I is the time at which processing of the last operation of the job is completed [Conway et. al. 1967]. Flow-time of job I is the total time that the job spends in the shop [Conway et. al. 1967].

To evaluate schedules the following *performance measures* or *optimality criteria* [Blazewicz2007]:

- *schedule length (makespan)*

$$C_{max} = max\{C_j\},$$

- *mean flow time*

$$\overline{F} = \frac{1}{n}\sum_{j=1}^{n} F_j ,$$

or mean weighted flow time

$$\overline{F_w} = \sum w_j Fj / \sum_{j=1}^{n} w_j ,$$

- *maximum lateness*

$$L_{\max} = \max\{L_j\}.$$

or other related criteria.

A schedule for which the value of a particular performance measure γ is at its minimum will be called *optimal*, and the corresponding value of γ will be denoted by γ^* [Blazewicz2007].

A *scheduling problem* Π is defined as a set of parameters not all of which have numerical values, together with an optimality criterion. An *instance I* of problem Π is obtained by specifying particular values for all the problem parameters [Blazewicz2007].

A *scheduling algorithm* is an algorithm which constructs a schedule for a given problem Π.

The theory of scheduling is characterized by a virtually unlimited number of problem types [Brucker2007]. In order to cope with this variety of scheduling problems a notation composed of three fields $\alpha \mid \beta \mid \gamma$, where introduced [Brucker2007]:

- the first field α describes the processor environment,
- the second field β describes task and resource characteristics,
- the third field, γ, denotes an optimality criterion (performance measure).

1.2. Communication delay and multi-processor task

In recent years, with the rapid development of parallel and distributed systems, the constraint that imposes that each task may be executed on a single processor at a time may be relaxed. In this context the delays caused by the communication between tasks cannot be ignored. There are three models to describe the communication problem in the context of scheduling problems.

In this respect, the task model from classical scheduling theory is enriched in order to incorporate the delays caused by communication. These delays may be handled implicitly or explicitly. In the first case, the communication times are already included in the task processing times. Usually, a task requires more than one processor at a time. Such a task is called multi-processor task. Multi-processor tasks may specify they processor requirements either in terms of simultaneously required processors, or in terms of an explicit specification of a processor subset (or processor subsets) which is or are required for processing. In the first case we will speak about *parallel processor requirement*, whereas in the second we will speak about *dedicated processor requirement* [Blazewicz2007].

1.3. Scheduling with limited processor availability

A machine system with limited availability is a set of machines (processors) which does not operate continuously; each machine is ready for processing only in certain time intervals of availability [Blazewicz2007]. We want to find a feasible schedule if one exists, such that all tasks can be processed within the given intervals of machine availability optimizing some performance criterion.

The term preemption is used as defined before. Often the notion of resumability is used instead of preemption. Under a resumable scenario a task may be interrupted when a machine becomes unavailable and resumed as the machine becomes available again without any penalty. Under the non-resumable scenario task preemption is generally forbidden. The most general scenario is semi-resumability [Blazewicz2007].

1.4. Scheduling under resource constraints

The scheduling model consider next is more complicated than the previous ones, because any task, besides processors, may require for its processing some additional scarce resources.

Resources, depending on their nature, may be classified into types and categories [Blazewicz2007]. The classification into *types* takes into account only the functions resources fulfill: resources of the same type are assumed to fulfill the same functions [Blazewicz2007]. The classification into *categories* will concern two points of view. First, we differentiate three categories of resources from the viewpoint of resource constraints. We will call a resource *renewable*, if only its total usage, i.e. temporary availability at every moment, is constrained. A resource is called *non-renewable*, if only its total consumption, i.e. integral availability up to any given moment, is constrained (in other words this resource once used by some task cannot be assigned to any other task). A resource is called *doubly constrained*, if both total usage and total consumption are constrained. Secondly, we distinguish two resource categories from the viewpoint of resource divisibility: *discrete* (i.e. discretely-divisible) and *continuous* (i.e. continuously-divisible) resources. In other words, by a discrete resource we will understand a resource which can be allocated to tasks in discrete amounts from a given finite set of possible allocations, which in particular may consist of one element only. Continuous resources, on the other hand, can be allocated in arbitrary, a priori unknown, amounts from given intervals [Blazewicz2007].

1.5. Multicriteria scheduling problem

Many scheduling problems in the production or service domains involve several criteria. Examples of such problems are time/cost trade-off problems. As a general rule, taking several criteria into account enables to provide the decision maker with a more realistic solution.

A multicriteria scheduling problem is a problem which consists of computing a Pareto optimal schedule for several conflicting criteria. This problem can be broken down into three sub-problems [T'kindt&Billaut2006]:

- modelling of the problem, whose resolution leads to the determination of the nature of the scheduling problem under consideration as well as the definition of the criteria to be taken into account,
- taking into account of criteria, whose resolution leads to indication of the resolution context and the way in which we want to take into account the criteria. The analyst

finalises a decision aid module for the multicriteria problem, also called a module for taking account of criteria,

- scheduling, whose resolution leads us to find a solution of the problem. The analyst finalises an algorithm for solving the scheduling problem, also called a resolution module for the scheduling problem.

At the phase of *taking account of criteria,* and following the information which it sets out, the analyst chooses a resolution approach for the scheduling problem and thus defines a scheduling problem. Taking account of the diversity of the methods of determining Pareto optima, the functions to optimise for the scheduling problem can take different forms. Each one translates a method of determining a Pareto optimum. The criteria do not change and they correspond to those defined during the phase of *modelling of the problem* [T'kindt&Billaut2006].

A multicriteria scheduling problem, after the modelling phase, can be noted in a general way by using the three-field notation, where the field γ contains the list of criteria: $\alpha \mid \beta \mid Z_1, Z_2, \ldots, Z_k$. The scheduling problem produced by the phase of taking account of the criteria may equally be noted by means of the three fields, where only field γ is spread [T'kindt&Billaut2006].

2. Complexity of scheduling problems

The aim of this chapter is to show the complexity of the scheduling problems in general and to presents the problem hierarchy describing the relationships between various scheduling problems.

Complexity theory is an important tool in scheduling research [Leung2004]. It provides a mathematical framework in which computational problems are studied so that they can be classified as “easy” or “hard” [Brucker2007]. This classification is useful in order to see if there is an efficient algorithm, especially in terms of time, for solving a particular problem. A problem belongs to a class of complexity, which informs us of the complexity of the "best algorithm" able to solve it. Hence, if a given problem is shown to belong to the class of "easy" problems then it means that we are able to exhibit a polynomial time algorithm to solve it. Usually this is good news but unfortunately this does not often happen for complex problems. Accordingly, if a problem belongs to the class of hard problems, it cannot be solved in polynomial time which, said differently, implies that for some instances the required CPU time to solve it becomes "exponential" [T’kindt&Billaut2006].

2.1. Problems, algorithms and complexity

A problem Π is described by giving [Garey&Johnson1979]:

- a general description of all its parameters, and
- a statement of what properties the answer, or solution, is required to satisfy.

An instance I of a problem Π is obtained by specifying particular values for all the problem parameters [Garey&Johnson1979]. With each instance there is a “size” associated. The size of an instance refers to the length of the data string necessary to specify the instance and it depends on the magnitude of the largest element [T’kindt&Billaut2006]. It is also referred to as the length (size) of the encoding scheme [Pinedo2008]. An encoding scheme maps problem instances into the strings describing them [Garey&Johnson1979].

Algorithms are general, step-by-step procedures for solving problems. An algorithm solves a problem Π if it finds a solution for any instance *I* of Π [Blazewicz et. al. 2007].

In general, we are interested in finding the most "efficient" algorithm for solving a problem. In its broadest sense, the notion of efficiency involves all the various computing resources needed for executing an algorithm. However, by the "most efficient" algorithm one normally means the fastest. Since time requirements are often a dominant factor determining whether or not a particular algorithm is efficient enough to be useful in practice, this single resource will be considered in the context of algorithm complexity analysis [Garey&Johnson1979].

The running time of an algorithm is measured by the number of basic computational steps it takes [Leung2004]. In order to define a computational step, a standard model of computing is used, the *Turing* machine. Any standard text on computational complexity contains the assumptions of the Turing machine [Pinedo2008].

Formally, the time complexity function of an algorithm *A* solving a problem Π is a function that maps each input length of an instance *I* of Π into a maximal number of elementary steps (or time unit) of a computer, which are needed to solve an instance of that size by algorithm *A* [Blazewicz et. al. 2007]. This function is not well-defined until one fixes [Garey&Johnson1979]:

- the encoding scheme to be used for determining input length (size) and
- the computer or computer model to be used for determining execution time of basic steps.

Different algorithms possess a wide variety of different time complexity functions, and the characterization of which of these are "efficient enough" and which are "too inefficient" will always depend on the situation at hand. However, computer scientists recognize a simple distinction that offers considerable insight into these matters. This is the distinction between polynomial time algorithms and exponential time algorithms [Garey&Johnson1979].

As it was aforementioned the efficiency of an algorithm is measured by an upper bound $T(n)$ on the number of computational steps that the algorithm takes in order to solve an instance *I* of a problem Π [Brucker2007]. In other words the efficiency of an algorithm for a given problem is measured by the maximum (worst-case) number of computational steps needed to obtain an optimal solution as a function of the size of the instance [Pinedo2008].

In most cases it will be difficult to calculate the precise form of *T*. For these reasons the precise form of *T* is replaced by its asymptotic order. Therefore, it is said that $T(n) \in O(g(n))$ if there exist constants $c > 0$ and a nonnegative integer n_0 such that $T(n) \leq cg(n)$ for all integers $n \geq n_0$ [Brucker2007].

A polynomial time algorithm is defined to be one whose time complexity function is O(g(n)) for some polynomial function g, where n is used to denote the input length [Garey&Johnson1979]. Any algorithm whose time complexity function cannot be so bounded is called an exponential time algorithm (although it should be noted that this definition includes certain non-polynomial time complexity functions, like $n^{\log n}$, which are not normally regarded as exponential functions) [Garey&Johnson1979].

The distinction between these two types of algorithms has particular significance when considering the solution of large problem instances. There is wide agreement that a problem has not been "well-solved" until a polynomial time algorithm is known for it. Hence, a problem is called intractable if it is so hard that no polynomial time algorithm can possibly solve it [Garey&Johnson1979].

This definition of "intractable" provides a theoretical framework of considerable generality and power. The intractability of a problem turns out to be essentially independent of the particular encoding scheme and computer model used for determining time complexity [Garey&Johnson1979] if “reasonable” encoding scheme and “reasonable” computer model are used.

A “reasonable” encoding scheme is one which satisfies the following two conditions:

- the encoding of an instance I should be concise and not “padded" with unnecessary information or symbols, and
- numbers occurring in I should be represented in binary (or decimal, or octal, or in any fixed base other than 1),

while a “reasonable” computer model is one in which there is a polynomial bound on the amount of work that can be done in a single unit of time (thus, for example, a model having the capability of performing arbitrarily many operations in parallel would not be considered "reasonable," and indeed no existing (or planned) computer has this capability.) [Garey&Johnson1979].

The definition of intractability allowed to distinguish between two different causes. The first, which is the one we usually have in mind, is that the problem is so difficult that an exponential amount of time is needed to discover a solution. The second is that the solution itself is required to be so extensive that it cannot be described with an expression having length bounded by a polynomial function of the input length [Garey&Johnson1979]. Next, the attention will be on the first type of intractability (only problems for which the solution length is bounded by a polynomial function of the input length will be considered).

2.2. Polynomial reduction

As theoreticians continue to seek more powerful methods for proving problems intractable, parallel efforts focus on learning more about the ways in which various problems are interrelated with respect to their difficulty. The principal technique used for demonstrating that two problems are related is that of "reducing" one to the other, by giving a constructive transformation that maps any instance of the first problem into an equivalent instance of the second. Such a transformation provides the means for converting any algorithm that solves the second problem into a corresponding algorithm for solving the first problem [Garey&Johnson1979].

It is said that problem P reduces to problem P’ if for any instance of P an equivalent instance of P’ can be constructed. In complexity theory usually a more stringent notion is used. Problem P polynomially reduces to problem P’ if a polynomial time algorithm for P’ implies a polynomial time algorithm for P. Polynomial reducibility of P to P’ is denoted by $P \propto P'$. If it is known that if there does not exist a polynomial time algorithm for problem P, then there does not exist a polynomial time algorithm for problem P' either [Pinedo2008].

The notion of polynomial reduction is central to the theory of NP-hardness. The theory of NP-hardness applies to decision problems only [Leung2004]. A decision problem is a problem for which the answer is "yes" or "no". Since almost all of the scheduling problems are optimization problems, it seems that the theory of NP-hardness is of little use in scheduling theory. But, all optimization (maximization or minimization) problems can be converted into corresponding decision problems by providing an additional parameter ω , and simply asking whether there is a feasible solution such that the cost of the solution is less or equal (or greater or equal for maximization problems) than ω [Leung 2004].

A problem is called polynomially solvable if there exists a polynomial p such that T(n) ∈ O(p(n)) where n is the input length with respect to a "reasonable" encoding scheme, i.e. if there is a k such that T(n) ∈ O(nk). If for a problem T(n) is polynomial with respect to unary encoding then the problem is called pseudo-polynomial [Brucker2007].

The class of all decision problems which are polynomially solvable is denoted by P [Brucker 2007]. NP refers to the class of decision problems which have "succinct" certificates that can be verified in polynomial time. "Succinct" certificates are those whose size is bounded by a polynomial function of the size of the input [Leung2004].

A decision problem Q is said to be NP-complete [Leung2004] if:

- Q is in the NP-class and
- All problems in the NP-class are reducible to Q.

A problem is said to be NP-hard if it satisfies only the second condition from the above definition [Leung2004]. Not all problems within the NP-hard class are equally difficult. Some problems are more difficult than others. For example, it may be that a problem can be solved in polynomial time as a function of the size of the problem in unary encoding, while it cannot be solved in polynomial time as a function of the size of the problem in binary encoding. For other problems there may not exist polynomial time algorithms under either unary or binary encoding. The first class of problems are not as hard as the second class of problems. The problems in this first class are usually referred to as NP-hard in the ordinary sense or simply NP-hard. The algorithms for this class of problems are called pseudo-polynomial. The second class of problems are usually referred to as strongly NP-hard [Pinedo2008].

To show a problem to be NP-complete, one needs to show that all problems in the NP-class are reducible to it. Since there are infinite number of problems in NP-class, it is not clear how one can prove any problem to be NP-complete. Fortunately, Cook [Cook1971] gave a proof that satisfiability problem is NP-complete, by giving a generic reduction from Turing machines to satisfiability [Leung2004]. From satisfiability problem other problems can be shown to be NP-complete by reducing it to the target problems. Because reducibility is transitive this is equal to showing that all problems in the NP-class are reducible to the target problems. Starting from satisfiability, Karp [Karp1972] showed a large number of combinatorial problems to be NP-complete [Leung2004].

Satisfiability problem is defined as follows: given a set of variables and a collection of clauses defined over the variables, is there an assignment of values to the variables for which each one of the clauses is true? [Pinedo2008]. The problem in which each clause contains exactly 3 literals is called the 3-satisfiability problem (3-SAT) [Brucker2007].

The diagram of Figure 2.1 shows some basic polynomial transformations between some problems. An arc from P to Q in Figure 2.1 indicates that P $\propto$ Q. Because all problems in Figure 2.1 belong to NP all these problem are NP-complete [Brucker2007].

$$\begin{tikzcd}
 & & \text{SAT} \arrow[d] & & \\
 & & \text{3-SAT} \arrow[dll] \arrow[dr] & & \\
\text{3DM} \arrow[d] & & & \text{VC} \arrow[dl] \arrow[dr] & \\
\text{PART} & & \text{HC} & & \text{CLIQUE}
\end{tikzcd}$$

Figure 2.1 Basic polynomial transformations [Brucker2007].

The problems of partition (PART), Hamiltonian circuit (HC) and clique (CLICUE) are very important for scheduling theory because problems of which the complexity is established through a reduction from partition typically allow for pseudo-polynomial time algorithms and are therefore NP-hard in the ordinary sense. NP-hard problems of which the complexity is established via a reduction from satisfiability, 3-partition, hamiltonian circuit or clique are strongly NP-hard [Pinedo2008].

2.3. Complexity hierarchy

To calculate the complexity of scheduling problems, a certain amount of traditional results exist in the literature. These results show the links between different single criterion deterministic scheduling problems. If a scheduling problem reduces to another scheduling problem, an algorithm for one scheduling problem can be applied to another scheduling problem as well [T'kindt&Billaut2006].

A considerable effort has been made to establish a problem hierarchy describing the relationships between the hundreds of scheduling problems. In the comparisons between the complexities of the different scheduling problems it is of interest to know how a change in a single element in the classification of a problem affects its complexity. In Figure 2.2 to 2.4 a number of graphs are exhibited that help determine the complexity hierarchy of deterministic scheduling problems [Pinedo2008]. These graphs illustrates polynomial reductions between scheduling problems. The vertices characterize the problems and where there is an arc between a vertex A and vertex B if A $\propto B$ [T'kindt&Billaut2006].

Such graphs exist for types of problems (figure 2.2), types of constraints (figure 2.3) and criteria (figure 2.4) [T'kindt&Billaut2006]:

- in figure 2.2, the presence of an arc from A towards B means that a polynomial reduction exists from an $A|\beta|\gamma$ problem towards the corresponding B$|\beta|\gamma$ problem.
- in figure 2.3, the presence of an arc from A towards B means that a polynomial reduction exists from the $\alpha|A|\gamma$ problem towards the corresponding $\alpha|B|\gamma$ problem.
- in figure 2.4 the presence of an arc from A towards B means that a polynomial reduction exists from the a $\alpha|\beta|A$ problem towards the corresponding $\alpha|\beta|B$ problem.

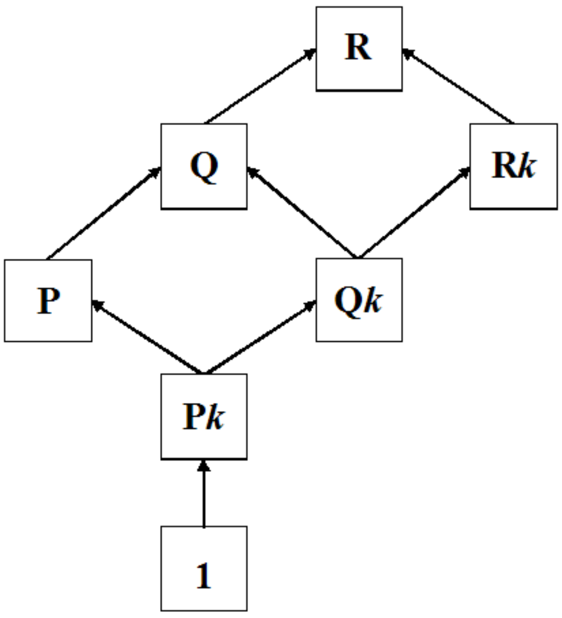


Figure 2.2 Reduction graph for processor characteristics [Blazewicz et.al. 2007]

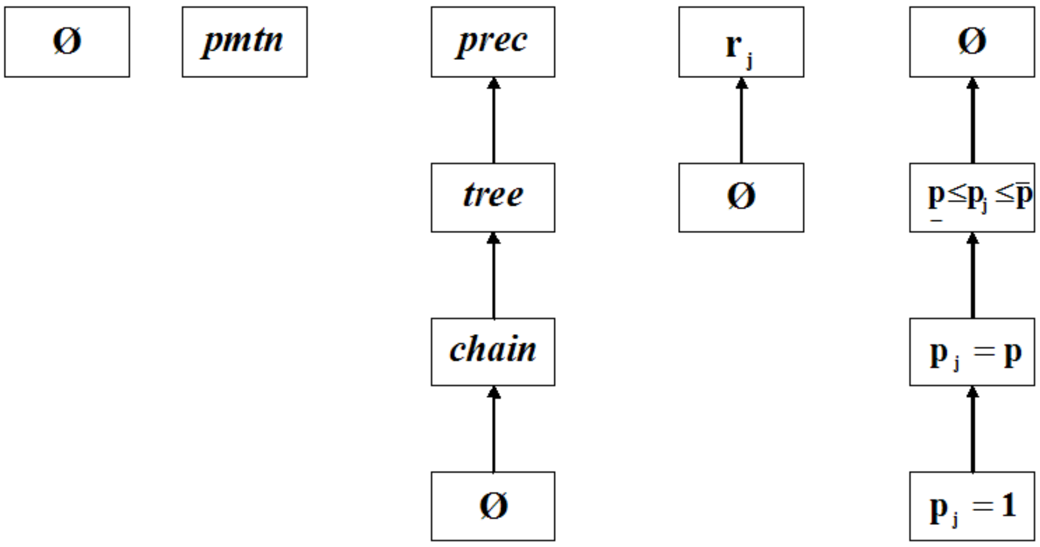


Figure 2.3 Reduction graph for task and resource characteristics [Blazewicz et.al. 2007]

$$\begin{tikzcd}
\overline{F}_w \arrow[r] & \overline{D}_w, \overline{E}_w & U_w \\
\overline{F} \arrow[u] \arrow[r] & \overline{D}, \overline{E} \arrow[u] & U \arrow[u] \\
 & L_{max} \arrow[u] \arrow[ur] & \\
 & C_{max} \arrow[u] &
\end{tikzcd}$$

Figure 2.4 Reduction graph for optimality criteria [Blazewicz et.al. 2007]

The reduction graphs presented are usable only when we already know the complexity of certain scheduling problems. There has been made a considerable effort for determining the complexity results (polynomially solvable, pseudo-polynomially solvable and NP-hard) of several scheduling problems [Brucker2007].

3. Evolutionary computation

The aim of this chapter is to give an overview of the basic elements of an evolutionary algorithm (representation, fitness function, selection, crossover, mutation and parameters) and to present some representation of feasible schedules.

Scheduling problems are optimization problems. When we address a scheduling problem, we must always look for its complexity, since this determines the nature of the algorithm to implement. If the problem under consideration belongs to the class P , we know that an exact polynomial algorithm exists to solve it. In this case it is convenient to use or to perfect such an algorithm. By contrast, if the problem is NP-hard, two alternatives are possible. The first is to propose an approximated algorithm, therefore an heuristic one, which calculates in polynomial time a solution which is as close as possible to the optimal solution. The second is to propose an algorithm which calculates the optimal solution of the problem, but for which the maximal complexity is exponential. In this case, the challenge is to design an algorithm which can solve problems of the largest possible size [T'kindt&Billaut2006].

In this thesis the first alternative is addressed by using evolutionary computation techniques in order to propose approximation algorithm for scheduling problems.

3.1. Overview of evolutionary algorithms

Evolutionary computation (EC) refers to computer-based problem solving systems that use computational models of evolutionary processes, such as natural selection, survival of the fittest and reproduction, as the fundamental components of such computational systems [Engelbrecht2002].

Evolution via natural selection of a randomly chosen population of individuals can be thought of as a search through the space of possible chromosome values. In that sense, an evolutionary algorithm (EA) is a stochastic search for an optimal solution to a given problem [Engelbrecht2002].

Figure 3.1 is an outline for a simple evolutionary algorithm [Ahn2006].

```
Step 1. Initialization
        Generate initial population P at random or with prior knowledge
Step 2. Fitness evaluation
        Evaluate the fitness for all individuals in P
Step 3. Selection
        Select a set of promising candidates S from P
Step 4. Reproduction
        Step 4.1. Crossover (optional)
                Apply crossover to the mating pool S for generating a set of offspring O
        Step 4.2. Mutation (probabilistic)
                Apply mutation to the offspring set O for obtaining its perturbed set O’
Step 5. Replacement
        Replace the current population P
Step 6. Termination
        If the termination criteria are not met, go to Step 2
```

Figure 3.1 Pseudo-code for simple evolutionary algorithm

This simple evolutionary algorithm is more complex than it seems at first glance. There are five important decisions that factor into the design of the algorithm [Ashlock2006]: What data structure will you use? What fitness function will you use? What reproduction (crossover and mutation) operators will you use? How will you select parents from the population, and how will you insert children into the population? What termination condition will end your algorithm?

There are different EC paradigms [Engelbrecht2002]: Genetic algorithms (GAs), Evolutionary programming (EP), Evolution strategies (ESs), Genetic programming (GP), Differential evolution (DE), Cultural evolution (CE), Co-evolution (CoE). Further we will refer only to Genetic algorithms (GAs), Evolutionary programming (EP) and Evolution strategies (ESs).

Representation

As the structure of a solution varies from problem to problem, a solution of a particular problem can be represented in a number of ways. Usually, a search method is most efficient in dealing with a particular representation and is not so efficient in dealing with other representations. Thus, the choice of an efficient representation scheme depends not only on the underlying problem but also on the chosen search method. The efficiency and complexity of a search algorithm largely depends on how the solutions have been represented and how suitable the representation is in the context of the underlying search operators. In some cases, a difficult problem can be made simpler by suitably choosing a representation that works efficiently with a particular algorithm [DeJong1997].

Initial population

Evolutionary algorithms are stochastic, population-based search algorithms. Each EA therefore maintains a population of candidate solutions. The first step in applying an EA to solve an optimization problem is to generate an initial population. The standard way of generating an initial population is to assign a random value from the allowed domain to each of the genes of each chromosome. The goal of random selection is to ensure that the initial population is a uniform representation of the entire search space. If regions of the search space are not covered by the initial population, chances are that those parts will be neglected by the search process. The size of the initial population has consequences in terms of computational complexity and exploration abilities [Engelbrecht2002].

Fitness function

In the Darwinian model of evolution, individuals with the best characteristics have the best chance to survive and to reproduce. In order to determine the ability of an individual of an EA to survive, a mathematical function, called fitness function, is used to quantify how good the solution represented by a chromosome is. Fitness function has an important role in an evolutionary algorithm because the evolutionary operators usually make use of the fitness evaluation of chromosomes [Engelbrecht2002].

Selection

Selection is one of the main operators used in evolutionary algorithms. The primary objective of the selection operator is to *emphasize* better solutions in a population. This operator does not create any new solution, instead it selects relatively good solutions from a population and deletes the remaining, not-so-good, solutions [DeJong1997]. The identification of good or bad solutions in a population is usually accomplished according to a solution's fitness. The essential idea is that a solution having a better fitness must have a higher probability of selection. However, selection operators differ in the way the copies are assigned to better solutions. Some operators sort the population according to fitness and deterministically choose the best few solutions, whereas some operators assign a probability of selection to each solution according to fitness and make a copy using that probability distribution. [DeJong1997].

There exist a various selection operators like proportionate selection, tournament selection, ranking selection, etc. Selection operators are characterized by their *selective pressure*, also referred to as the *takeover time*, which relates to the time it requires to produce a uniform population. It is defined as the speed at which the best solution will occupy the entire population by repeated application of the selection operator alone. An operator with a high selective pressure decreases diversity in the population more rapidly than operators with a low selective pressure, which may lead to premature convergence to suboptimal solutions. A high selective pressure limits the exploration abilities of the population [Engelbrecht2002].

Reproduction (crossover and mutation)

Reproduction is the process of producing offspring from selected parents by applying crossover and/or mutation operators.

Crossover is the process of creating one or more new individuals through the combination of genetic material randomly selected from two or more parents. If selection focuses on the most-fit individuals, the selection pressure may cause premature convergence due to reduced diversity of the new populations [Engelbrecht2002].

Mutation is the process of randomly changing the values of genes in a chromosome. The main objective of mutation is to introduce new genetic material into the population, thereby increasing genetic diversity. [Engelbrecht2002].

Stopping criteria

The evolutionary operators are iteratively applied in an EA until a stopping condition is satisfied. The simplest stopping condition is to limit the number of generations that the EA is allowed to execute, or alternatively, a limit is placed on the number of fitness function evaluations. This limit should not be too small, otherwise the EA will not have sufficient time to explore the search space [Engelbrecht2002].

In addition to a limit on execution time, a convergence criterion is usually used to detect if the population has converged. Convergence is loosely defined as the event when the population becomes stagnant. In other words, when there is no genotypic or phenotypic change in the population [Engelbrecht2002]:

3.2. Classification of parameter control techniques

The issue of setting the values of various parameters of an evolutionary algorithm (EA) is crucial for good performance. In classifying parameter control techniques of an evolutionary algorithm, many aspects can be taken into account [Siarry&Michalewicz2008]:

- *What* is changed (e.g., representation, evaluation function, operators, selection process, mutation rate, population size, and so on)?
- *How* the change is made (i.e., deterministic heuristic, feedback-based heuristic, or self-adaptive)?
- *The evidence* upon which the change is carried out (e.g., monitoring performance of operators, diversity of the population, and so on)?

To classify parameter control techniques from the perspective of what component or parameter is changed [Siarry&Michalewicz2008], it is necessary to agree on a list of all major components of an evolutionary algorithm, which is a difficult task in itself: representation of individuals, evaluation function, variation operators and their probabilities, selection operator (parent selection or mating selection), replacement operator (survival selection or environmental selection), population (size, topology, etc.).

Methods for changing the value of a parameter (i.e., the "how-aspect") can be classified into [Siarry&Michalewicz2008]: *parameter tuning* and *parameter control*. By parameter tuning we mean the commonly practiced approach that amounts to finding good values for the parameters *before* the run of the algorithm and then running the algorithm using these values, which remain fixed during the run. Parameter control forms an alternative, as it amounts to starting a run with initial parameter values that are changed *during* the run. We can further classify parameter control into one of the three following categories [Siarry&Michalewicz2008]: deterministic, adaptive and self-adaptive. This terminology leads to the taxonomy illustrated in Figure 3.1.

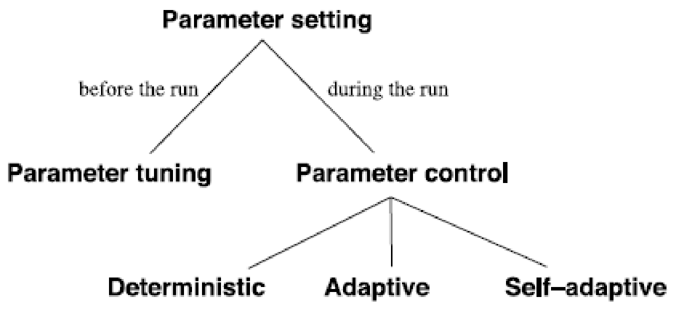


Figure 3.1 Global taxonomy of parameter setting in EAs [Siarry&Michalewicz2008]

3.3. Evolutionary computation vs. classical optimization

While classical optimization algorithms have been shown to be very successful (and more efficient than EAs) in linear, quadratic, strongly convex, unimodal and other specialized problems, EAs have been shown to be more efficient for discontinuous, non-differentiable, multimodal and noisy problems.EC and classical optimization (CO) differ mainly in the search process and information about the search space used to guide the search process [Engelbrecht2002]:

- *The search process*: CO uses deterministic rules to move from one point in the search space to the next point. EC, on the other hand, uses probabilistic transition rules. Also, EC applies a parallel search of the search space, while CO uses a sequential search. An EA search starts from a diverse set of initial points, which allows parallel search of a large area of the search space. CO starts from one point, successively adjusting this point to move toward the optimum.
- *Search surface information:* CO uses derivative information, usually first order or second-order, of the search space to guide the path to the optimum. EC, on the other hand, uses no derivative information. The fitness values of individuals are used to guide the search.

According to the no-free-lunch (NFL) theorem [Wolpert&Macready 1996]. there cannot exist any algorithm for solving all (e.g. optimization) problems that is generally (on average) superior to any competitor, the question of whether evolutionary algorithms (EAs) are inferior/superior to any alternative approach is senseless. What could be claimed solely is that EAs behave better than other methods with respect to solving a specific class of problems—with the consequence that they behave worse for other problem classes [DeJong1997, Baeck. et. al 2000].

Part B - Contributions

4. Uniform parallel machines scheduling problem

The aim of this chapter is to present a new genetic algorithm for the uniform parallel machines scheduling problem. The proposed algorithm not only obtains better results than other algorithm it also computes the result faster.

Uniform parallel machines are a special class of resources [Blazewicz et. al. 2007] in which machines have different speeds but the speed is constant and does not depends on tasks. Because the problem was proven to be NP-hard [Garey&Johnson1979] we propose a new genetic algorithm (GASP) in order to find a solution of this problem [Mihăilă&Mihăilă2008a]. We report some results and the performance of the presented approach are compared with other optimization techniques. Empirical results indicate that GASP is more efficient [Mihăilă&Mihăilă2008b].

4.1. Introduction

Scheduling is known to be NP-hard, therefore the use of heuristics is the de-facto approach in order to cope in practice with its difficulty. Beside heuristic approaches like local search [Ritchie&Levine2003], simulated annealing [Abraham et. al. 2000] [Yarkhan&Dongarra2002], tabu search [Abraham et. al. 2000] and genetic algorithms [Abraham et. al. 2000][Zomaya&The2001] were also used for scheduling problems. Ritchie and Levine [Ritchie&Levine2004] combined an ant colony optimization algorithm with a tabu search algorithm for the problem while Ye et al. [Guangchang et. al. 2006] formulated a multi-objective optimization approach to simultaneously optimize the completion time and the total execution cost. Other approaches for the problem include particle swarm optimization [Abraham et. al. 2006], fuzzy based scheduling [Kumar et. al. 2004] and economic-based approaches [Buyya et. al. 2000].

4.2. Uniform parallel machines scheduling problem

Formally, our scheduling problem, can be described as follows: n independent tasks $T = \{T_1, T_2, ..., T_n\}$ must be allocated to m uniform parallel machines $M = \{M_1, M_2, ..., M_m\}$ considering the objective of minimizing the completion time and utilizing the resources effectively. The speed of each machine is expressed in number of cycles per unit time, and the length of each task in number of cycles. Each task T_i has processing requirement P_i cycles and machines M_k has speed of S_k cycles/second. Each task T_i has to be processed on machine M_k, until completion [Grosan et. al. 2007].

The objective of our scheduling problem is to minimize makespan of the obtained schedule.

4.3. Genetic algorithm for scheduling problem

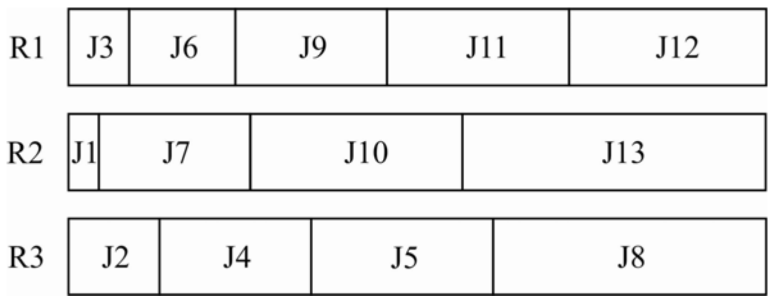
The algorithm starts with a population of randomly generated chromosomes (potentially solutions). In order to generate a new population we apply the following genetic operators [Baeck et. al. 2000]: binary tournament selection, for parent selection, and one-point crossover and gene mutation for generating new offspring chromosomes that will form a new population. The evolution process is similar to the evolution scheme of a standard genetic algorithm. We additionally used an elitism selection.

The solution to the scheduling problem is represented in the evolutionary algorithm as a string (chromosome) of length equal to the number of tasks. The value corresponding to each position i in the string represents the machine to which task i was allocated.

If we consider the case of 13 tasks and 3 machines then a chromosome of the task allocation can be represented as follows:

2	3	1	3	3	1	2	3	1	2	1	1	2

The corresponding Gantt chart of the encoded scheduling is:



4.4. Experimental results

In order to test our algorithm (GASP) we have conducted several experiments using four test instances and we have compared the obtained results with other optimization techniques: Genetic Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Multi-Objective Evolutionary Algorithm (MOEA).

Each experiment was repeated 10 times with different random seeds and each run was conducted with the following parameter settings: population size: 20; number of iterations: 50 * *m* * *n*; crossover probability: 0.45 (for instance 1) and 0.18 (for instance 2, 3, 4 and 5); mutation probability: 0.35 (for instance 1) and 0.02 (for instances 2, 3, 4 and 5).

The average makespans and the standard deviation reported for 10 trials are illustrated in Table 4.1.

Table 4.1. Average makespan and standard deviation.

Instance	Optimal result	Average makespan	Standard deviation
1	46	46	0
2	85.5279	85.5431	0.009
3	41.5788	41.7395	0.0856
4	35.1303	35.3785	0.0477
5	59.1658	59.3041	0.0461

Figure 4.1 illustrates, for instances 1, 2, 3, 4 and 5, the optimum, the best and average results (makespans) obtained by GASP.

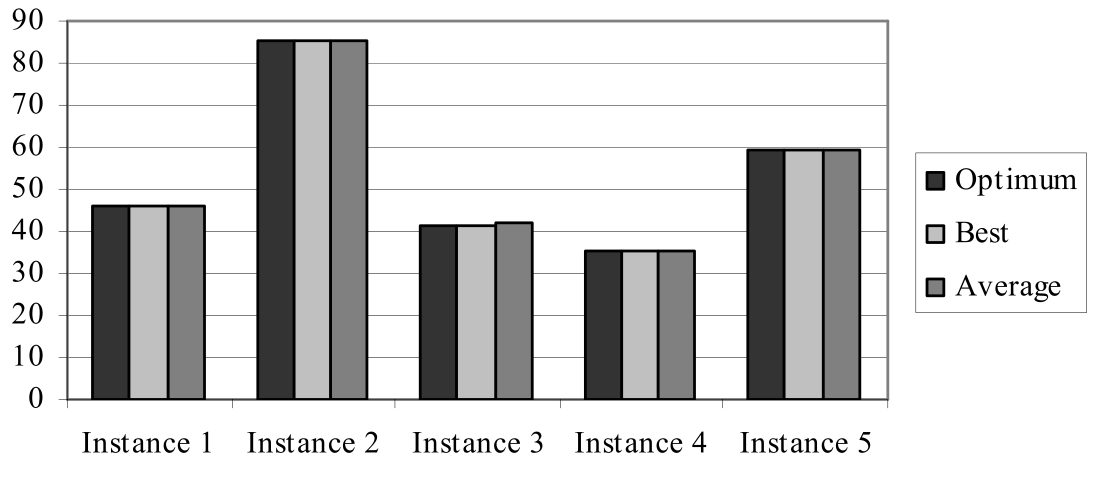


Figure 4.1. Best and average results obtained by GASP for instances 1, 2, 3, 4 and 5.

We compared the results obtained by our Evolutionary Algorithm for Scheduling Problems (GASP) with other techniques used for scheduling optimization: Genetic algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO).

The average makespan values [Abraham et. al. 2008] and standard deviations reported for 10 trials are illustrated in Table 4.2, where *am* stands for average makespan and *sd* for standard deviation (for PSO instance 2 we consider that there is a typing error). The results obtained by considered algorithm for comparison were taken from [Abraham et. al. 2006][Abraham et. al. 2008]. As can be seen from Table 4.2 the GASP algorithm gave the best result for all considered instances. We have conducted another experiment in order to see if the performance holds in case of reduction of the number of iterations. We tested the algorithm for 25**m***n*, GASP (2), and for *m***n*, GASP (3), number of iterations. The other parameters setting remained unchanged.

Table 4.2. Performance comparison.

Instance		1	2	3	4	5
Optimum		**46**	**85.5279**	**41.5788**	**35.1303**	**59.1658**
GA	*Am*	47.1167	85.7431	42.927	38.0428	
	Sd	±0.7700	±0.6217	±0.415	±0.6613	
SA	*Am*	46.6	90.7338	55.4594	41.7889	
	Sd	±0.4856	±6.3833	±2.0605	±8.0773	
PSO	*Am*	46.2667	84.0544	41.9489	37.6668	
	Sd	±0.2854	±0.5030	±0.6944	±0.6068	
ACO	*Am*	46.2667	88.1575			
	Sd	±0.2854	±0.6423			
GASP (1)	*Am*	**46**	**85.5431**	**41.7395**	**35.3785**	**59.3041**
	Sd	±0	±0.0090	±0.0856	±0.0477	±0.0461
GASP (2)	*Am*	**46.05**	**85.5595**	**41.8042**	**35.6098**	59.3625
	Sd	±0.15	±0.0092	±0.0775	±0.1398	±0.0716
GASP (3)	*Am*	46.5667	**85.681**	42.3229	**36.455**	59.7058
	Sd	±0.3512	±0.0803	±0.3036	±0.6077	±0.1683

As can be seen from Table 4.2, the performance of GASP holds even if we halve the number of iterations. This means that the GASP algorithm obtains very good results much faster than the considered algorithms. For the case of *m***n* number of iterations, the GASP gave very good results compared with other techniques taking into account the fact that other techniques used 50**m***n* number of iterations.

We also compared our algorithm (GASP) with Multi-Objective Evolutionary Algorithm (MOEA) [Abraham et. al. 2008]. The average result (makespan) for 10 runs is presented in Table 4.3. Results for MOEA were taken from [Abraham et. al. 2008].

Table 4.3. Performance comparison with MOEA

Instance		1	2	3	4	5
Optimum		**46**	**85.53**	**41.58**	**35.13**	**59.17**
MOEA	*am*	**46**			36.68	
GASP (1)	*am*	**46**	85.55	41.74	**35.38**	**59.37**
	sd	0	0.0084	0.055	0.067	0.0659
GASP (2)	*am*	46.15	85.57	41.8	**35.67**	**59.51**
	sd	0.2291	0.0108	0.074	0.2011	0.1328
GASP (3)	*am*	46.48	85.65	42.27	**35.94**	**59.84**
	sd	0.3609	0.0484	0.29	0.3032	0.2112

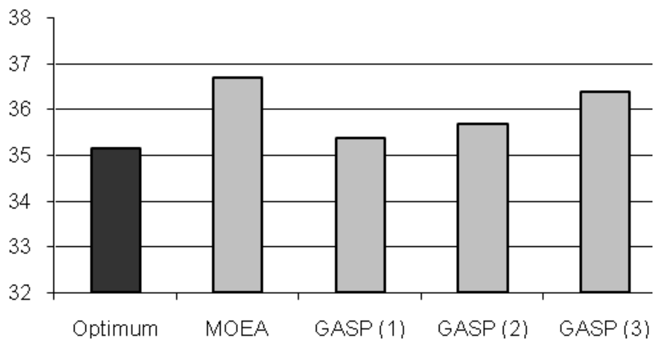


Figure 4.2 Average makespan of MOEA and GASP for test instances 4.

As can be seen from Table 4.3 GASP gave good results even when the population size and the number of iterations were halved. GASP (3) also gave good results taking into account the values of population size and number of iterations. We mention that both GASP (2) and GASP (3) have found the optimum value for instance 1. Figure 4.2 illustrates the average makespan, for instance 4, of MOEA and GASP (with 3 parameters setting) in 10 trials.

4.5. Conclusions and further research

From the data reported above we can conclude that GASP has given excellent results when compared to other techniques. Even though the GASP approach obtained better results for the considered test problems, compared with the results obtained by other optimization techniques, more conclusions could be drawn only after extensive validation using bigger problems.

Our future research plan is to extend the approach for scheduling problems involving unrelated parallel machines and dedicated machines and to test our algorithms with real data.

5. Permutation flow shop scheduling problem

The aim of this chapter is to present a new hybrid genetic algorithm for permutation flow shop scheduling problem. The novelty of the proposed algorithm consists in using a combination of a random initialization procedure and an initialization procedure based on NEH construction heuristic, in defining a new crossover operator and in using a mutation operator defined as a combination between an operator based on NEH construction heuristic and shift mutation. The results obtained by the proposed algorithm are comparable with the results obtained by other algorithms.

The objective of the permutation flow shop scheduling problem is to find a job sequence that will minimize a given criterion knowing that all jobs have the same processing order on machines. Because the problem is NP-hard, many heuristic and meta-heuristic methods have been proposed. The proposed algorithm [Mihăilă et.al.2008b] uses the NEH constructive heuristic for generating a predefined percent of the chromosomes of initial population in order to raise and speed up the chance of finding a good solution. The NEH constructive heuristics is also used by mutation operator in order to improve the obtained chromosomes. The results obtained by the proposed genetic algorithm are compared against the best results reported by an iterated greedy algorithm.

5.1. Introduction

The permutation flow shop scheduling problem (PFSP) was first studied by Johnson in 1954 [Johnson1954] and since then many heuristic and meta-heuristic methods have been proposed [Ruiz&Stützle2007]. The heuristic methods range from constructing heuristics like Rapid Access [Dannenbring1977] or NEH [Nawaz et. al. 1983] to improvement heuristics like RACS and RAES [Dannenbring1977] or that proposed by Suliman [Suliman2000]. The meta-heuristic methods can also be viewed as improvement heuristics. These methods range from those that improve a given schedule, like Tabu Search [Grabowski&Wodecki2004], [Nowicki&Smutnicki1996], [Taillard1990] Simulated Annealing [Osman&Potts1989] or

Iterative Greedy [Ruiz&Stützle2007], to those working with a collection of schedules, like Genetic Algorithm [Chen et. al. 1995] [Murata et. al. 1996], [Reeves&Yamada1998], [Ruiz et. al. 2004]or Ant Colony [Rajendran&Ziegler2004].

5.2. Permutation flow shop scheduling problem

As we have already mentioned the objective of the permutation flow shop scheduling problem is to find a sequence for processing a set of *n* jobs, $J = \{J_1, ..., J_n\}$ on a set of *m* processors or machines, P = $\{P_1, ... , P_m\}$ such that a given criterion is optimized. The criterion used in this chapter is the total idle time accumulated on the last machine and the objective is to minimize this criterion.

Each job J_i, $i = 1, ..., n$, is composed of a set of *m* tasks and each task *k*, $k = 1, ..., m$, has to be executed on a different machine, i.e. in order to be complete, a job has to be processed on each machine. The processing time of task *k* of job J_i is denoted by $p_{i,j}$.

In permutation flow shop problem all jobs have same processing order on machines and therefore once the job sequence on the first machines is fixed it will be kept on all remaining machines [Blazewicz et. al. 2007]. While the machine sequence of all jobs is the same, the problem is to find the job sequence that will minimize the given criterion [Blazewicz et. al. 2007], in our case the total idle time accumulated on the last machines.

The permutation flow shop scheduling problem, as a particular case of the flow shop scheduling problem fulfills several assumption that are commonly made regarding the flow scheduling problem [Ruiz&Maroto2004]: each machine can handle only one job at a time [Blazewicz et. al. 2007]; each job can be performed only on one machine at a time [Blazewicz et. al. 2007]; there are no precedence constraints among tasks of different jobs [Blazewicz et. al. 2007]; all jobs are available for processing at time 0 [Ruiz&Maroto2004]; the set-up times of the jobs on machines are negligible and therefore can be ignored [Ruiz&Maroto2004]; no preemption is allowed, i.e. the processing of a job on a machine cannot be interrupted [Ruiz&Maroto2004]; the machines are continuously available [Ruiz&Maroto2004]; in-process inventory is allowed, i.e. if the next machine on the sequence needed by a job is not available, the job can wait and joins the queue of that machines [Ruiz&Maroto2004].

In this chapter the permutation flow shop scheduling problems which comply with this assumption are considered. Since the considered problem is known to be NP-hard [Garey&Johnson1979] a meta-heuristic method was chosen in order to solve it.

5.3. Proposed genetic algorithm

In order to find a solution (schedule) to the permutation flow shop scheduling problem, described in previous section, we used a generational genetic algorithm.

Genetic algorithms use a population (collection) of chromosome (encoded possible solution) which evolves through genetic operators to a new population. This process of evolution is guided by the fitness function which measure the goodness of chromosomes and is repeated by a predefined number of times until a given stopping criteria is satisfied. The best chromosome from the final population is reported as the output of the algorithm. The initial population is usually randomly generated.

The solution of PFSP problem is encoded as a string permutation of the jobs. Thus the chromosome has a length equal with the number of jobs and the value (gene) corresponding to each position in the string represents a job. The relative order of the jobs in the permutation indicates the processing order of the jobs by the machines [Ruiz et. al. 2004].

In order to evaluate the quality of a chromosome we use as a fitness function the total idle time accumulated on the last machine.

Initial population was generated using a combination of a random initialization procedure and an initialization procedure based on NEH construction heuristic [Nawaz et. al. 1983]. We inserted the schedule (chromosome) constructed with NEH heuristic into the initial population and we also kept it as a basis, called *init* hereafter, for constructing other schedules. Initialization procedure based on NEH construction heuristic uses the idea of destruction-construction phases introduced in [Ruiz&Stützle2007] for an iterated greedy algorithm. In order to generate a new schedule (chromosome) some (%*n*) jobs are randomly removed from the *init* base and reinserted using NEH construction heuristic. We insert the new scheduled (chromosome) into the initial population and we update the *init* base to the value of new constructed schedule.Random initialization procedure also uses *init* base for generating a new schedule but it does not update its value. In order to generate a new schedule (chromosome) we first cut the *init* base after a randomly generated cut point and swap the resulting two segments, and then we perform *n* swap of two jobs randomly determined. The initial population is formed of 70% individuals created using the random initialization procedure and 30% individuals generated using the initialization procedure based on NEH.

In order to select the chromosome for the crossover operator we used binary tournament selection [Back et. al. 2000] which consists in randomly picking two chromosomes from the current population and choosing the best one.

Additionally we used an elitist selection [Back et. al. 2000] in order to prevent the loss of best chromosomes obtained so far. In this respect 20% of best chromosomes of the current population were copied into the next (new) population.

The crossover operator used in order to produce new (offspring) chromosome can be described as follows:

- randomly generate one cut point
- copy, with a given probability, the first or the last segment from the parent chromosomes into offspring chromosomes, i.e. if the first segment of the first parent chromosome is copied as the first segment into the first offspring chromosome then the first segment of the second parent chromosome will be copied as first segment into the second offspring chromosome (the same tactic holds for the last segment)
- complete the remaining parts of the offspring chromosome, starting right after the previous copied segment until the end, with the missing genes from the opposite parent chromosomes, i.e. the first offspring chromosome will take the missing genes from the second parent chromosome while the second offspring chromosome will be completed with the genes from the first parent chromosome. If the first segment was copied previously then the parent chromosomes will be traversed from the cut point

to right and when the end of the chromosome is reached the traversing order will begin from the first gene and will continue until the cut point is found. If the last segment was copied previously then the traversing policy is reversed.

As a mutation operator we used a combination between an operator based on NEH construction heuristic, named NEH mutation hereafter, which is similar with the initialization procedure based on NEH construction heuristic and shift mutation [Ruiz et. al. 2004] operator which consists in randomly selecting a position of the chromosome and relocating the gene (job) corresponding to chosen position to another randomly selected position while the genes (jobs) between these two positions move along. The NEH mutation operator has the goal to improve the chromosomes obtained by the crossover operator while the shift mutation introduces new chromosomes in order to reduce the loss the diversity of population.

Taking into account the suggestion from [Ruiz et. al. 2004] we have modified the survival operator of the generational genetic algorithm in the sense that we inserted into the next (new) population only distinct chromosome in order limit the effect of premature convergence and to increase the populations' diversity.

5.4. Experimental results

In order to test the performance of our algorithm we used the standard benchmark set [Taillard1993] from which we have chosen the 10 instances of 50 jobs and 20 machines and 10 instances of 100 jobs and 20 machines. These instances were chosen because it has been proven that some of these instances are very difficult to solve[Ruiz&Stützle2007].

Each experiment (instance) was repeated 10 times with different random seeds. Specific parameter settings for each run are described in Table 5.1.

Table 5.1 Parameter settings

Parameter	Value
Population dimension	100
number of generations	1000
crossover probability	0.5
mutation probability	0.1

The best and average results and the lower and upper bounds indicate the total completion time of a schedule, known as makespan. In order to determine the makespan of our result we added the total idle time accumulated on the last machine (computed by the fitness function) to the total execution time of the last machine.

Figure 5.1 and Figure 5.2 graphically presents, for each considered instances, the best and average results compared against the best known upper bound.

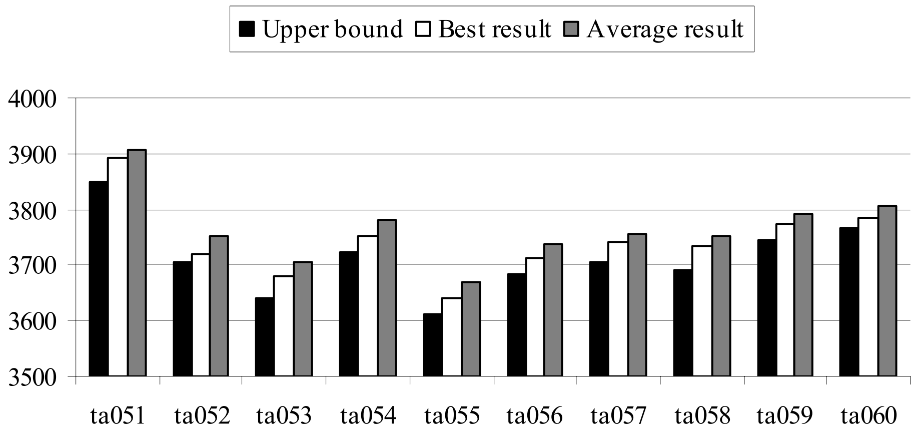


Figure 5.1 Best and average results for instances ta051-ta060

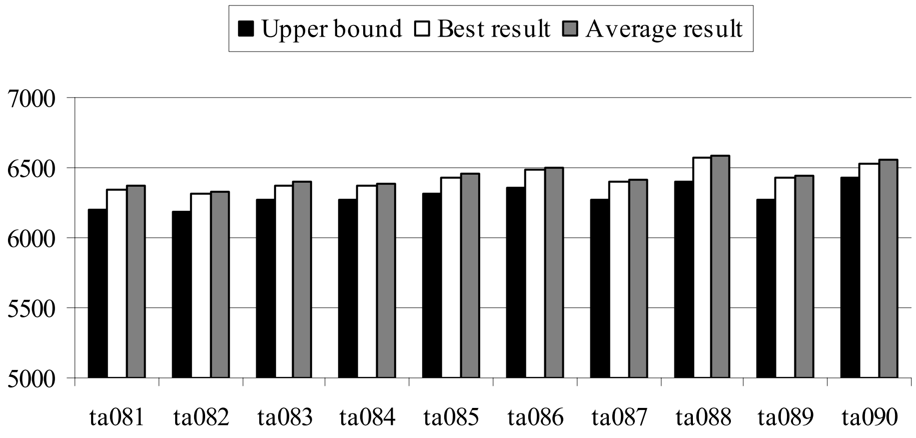


Figure 5.2 Best and average results for instances ta081-ta090

As can be seen from Figure 5.1 the results obtained are very close to upper bonds. The difference between best results and upper bounds are in the range 0.43% and 1.16% while the difference between average results and upper bound range from 1.05% to 1.74% for instances ta051-ta060.

We compared the obtained results with the result obtained by an iterated greedy algorithm with local search [Ruiz&Stützle2007]. The best results of the iterated greedy algorithm were taken from [Ruiz&Stützle2007]. The comparison of best results is illustrated in Figure 5.3 and Figure 5.4.

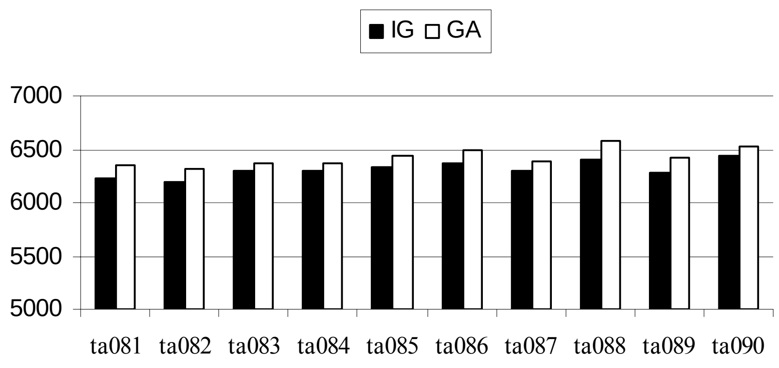


Figure 5.3 Comparison of best results for instances ta051-ta060

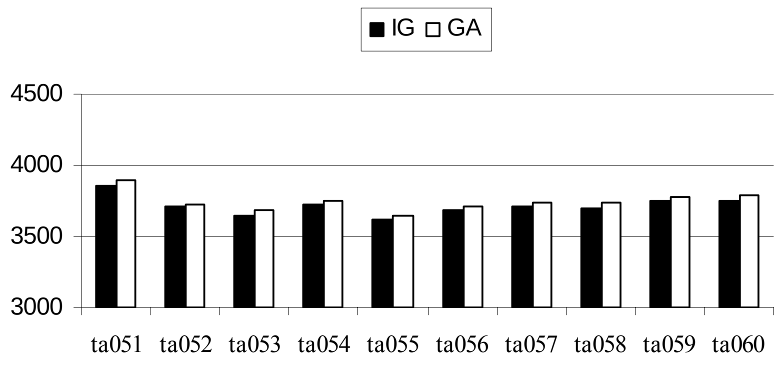


Figure 5.4 Comparison of best results for instances ta081-ta090

As can be seen from Figure 5.3 and Figure 5.4 the results obtained by the proposed genetic algorithm are very close to the results obtained by the iterated greedy algorithm. The difference between the best results of iterated greedy algorithm and the best results of genetic algorithm ranges from 0.40% to 1.08% for instances ta051-ta060 and 1.05% to 2.69% for instances ta081-ta090.

5.5. Conclusions and further research

In this chapter a genetic algorithm for solving permutation flow shop scheduling problems has been presented. The results obtained by the proposed algorithm are close to the result reported by other algorithms.

In future, other test instance from the standard benchmark set (e.g. 100 jobs and 20 machines, 200 jobs and 20 machines) will be considered for testing the performance of the proposed algorithm and the influence of different genetic operators, i.e. crossover and mutation, will be investigated.

6. Video-proxy cache scheduling problem

The aim of this chapter is to present two new ways of defining the utility of the objects stored inside a video proxy-cache and a new genetic algorithm used for determining the coefficients that appear in those two definitions with the aim of maximizing the byte hit rate. The obtained result by the proposed algorithm with one utility function is similar or even better than those obtained by other metric.

For video-proxy cache scheduling problem we introduce two new utility functions that consider the utility of the objects when performing cache replacement operations. The coefficients that fine-tune those functions are determined using a genetic algorithm [Mihăilă&Cobârzan2008]. Measurements regarding the efficiency in terms of byte hit rate when using those utility functions are made and the obtained results are compared with those yielded by classic cache replacement algorithms [Cobârzan&Mihăilă2008].

6.1. Introduction

The volume of multimedia and especially video content on the Internet has constantly increased in recent years. Due to the characteristics of this types of popular data (large sizes, limits on accepted latency during playback etc.) there is a lot of stress on the transport infrastructure which is in the general case the Internet. A proxy-cache is an entity which acts like an intermediary in a transaction in which a client requests a multimedia/video object. In such a case, the proxy-cache acting on behalf of the client, starts retrieving the object from an origin server and streams it towards the client. In the case a cache is also active, the content being streamed, or parts of it, can be saved locally.

There are numerous approaches to video caching: caching of a prefix in [Sen et. al. 1999], caching of a prefix and of selected frames in [HsiuMa&Du2000], caching of a prefix combined with periodic broadcast in [Yang& Towsley2002] or caching of hotspot segments in [Fabmi et. al. 2001]. Other proposals in the same category include caching of a prefix based on popularity [Park et. al. 2001], segment-based prefix caching [Wu et. al. 2001], and variable sized chunk caching [Balafoutis et. al. 2002]. There have been distributed approaches to the problem too: multiple video servers accessible via the web managing tertiary storage systems in [Brubeck&Brubeck1996] or cooperative caching video server in [Acharya2002] .

6.2. Proposed distributed video proxy-cache system

In [Cobârzan2005] a distributed system for video proxy-caching that is able to dynamically adjust the number of participating nodes in the federate cache depending on client request patterns and current load was introduced. Within the proposed system, cache replacement operations are performed with respect to the *utility value* of the objects, meaning that the objects with the smallest utility will be discarded when space has to be freed to accommodate new objects. The utility of an object is computed using a function u defined in [Cobârzan2005] as follows: $u : LC \to R$ (LC is the content of the local cache)

$$u(o) = coef_1 * size(o) + coef_2 * \frac{1}{timeLastAccess(o)} + coef_3 * hitCount(o) + coef_4 * qualityValue(o), \tag{6.1}$$

where:

- *size*(*o*) is the size of the object,
- *timeLastAccess*(*o*) indicates the last time the object has been requested,
- *hitCount*(*o*) shows the number of times the object has been served from the cache
- *qualityValue*(*o*) $\in$ [0..1] is the measure of the object's quality
- $coef_1, coef_2, coef_3, coef_4 \in [0, 1]$ and $coef_1 + coef_2 + coef_3 + coef_4 = 1$.

During the initial performance evaluation of the system we have noticed that when using the formula in defined in [Cobârzan2005] the impact of the *timeLastAccess* and *hitCount* is negligible due to the difference in magnitude order when compared to *size* and *hitCount*. In order to correct the observed negative aspects we introduce two new ways of defining the utility of an object:

$$\begin{aligned} u(o) = coef_1 * \frac{size(o)}{MSIZE} + coef_2 * \frac{timeLastAccess(o)}{MTLA} + \\ + coef_3 * \frac{hitCount(o)}{MHC} + coef_4 * qualityValue(o) \end{aligned} \tag{6.2}$$

and

$$u(o) = -\left[\begin{aligned} coef_1 * \frac{1}{size(o)} + coef_2 * \frac{1}{\sqrt[4]{timeLastAccess(o)}} + \\ + coef_3 * \frac{1}{hitCount(o)} + coef_4 * \frac{1}{qualityValue(o)} \end{aligned}\right] \tag{6.3}$$

where:

- *MSIZE* - the size of the largest stored object within the local cache *LC* until that moment;
- *MTLA* - the moment in time of the last request for an object in *LC*;

- *MHC* - the number of times the most popular object in *LC* has been requested.

Additionally $coef_1$, $coef_2$, $coef_3$, $coef_4 \in [0, 1]$ as well $coef_1 + coef_2 + coef_3 + coef_4 = 1$ still have to hold for both equation (6.2) and (6.3). We also have to note that the formulae (6.2) and (6.3) might require further tuning since we have not considered the quality value for objects during our measurements.

In [Cobârzan2005] we considered some fix values for $coef_1$ to $coef_4$. Our aim is to be able to provide an intelligent method of determining those coefficients so that different metrics are maximized/minimized (byte hit ratio, object hit ratio/latency). This has lead to our current approach of using genetic algorithms. As a first step we have focused on choosing the four values $coef_1$ to $coef_4$ in the utility formulae (6.2) and (6.3) based on the client request patterns (number of request for each object, moment in time when it is requested) as well as on objects characteristics (size distribution) so that the *byte hit ratio* is maximal.

6.3. Genetic algorithm for determining utility function coefficients

In order to find "good" values of coefficients for the utility functions defined in (6.2) and (6.3) we have used a genetic algorithm (GeCo) which starts with a population of randomly generated solutions (chromosomes). A new population is generated by applying the following genetic operators [Baeck et. al. 2000]: *binary tournament selection* for parent selection, and *convex crossover* and *radius gene mutation* for generating new offspring chromosomes that will form a new population. The evolution process is similar to the evolution scheme of a standard genetic algorithm. We additionally used an elitist selection.

The solution is represented as a string (chromosome) of length equal to the number of coefficients minus 1. We have considered only the first three coefficients disregarding the last one which has the value set to 0. For our problem this means that we are not interested in the quality of the requested/retrieved objects. The use of the forth coefficient makes sense only if transcoding operations are supported within the video proxy-cache. The value corresponding to each position i in the string represents the i^{th} coefficient. These values range from 0 to 1.

We have used both formulae (6.2) and (6.3) when computing the utility of an object. Before this we scale the gene values from the chromosome in order to assure that the sum of coefficients equals 1. Thus, the value of each coefficient is computed using the formula:

$$coef_i = \frac{gene_i}{\sum_{i=1}^{3} gene_i} \qquad (6.4)$$

where *chromosome* = ($gene_1$, $gene_2$, $gene_3$).

The quality evaluation of each solution (chromosome) was done by computing the *byte hit ratio*. Our goal was to maximize this value.

6.4. Experimental results

The GeCo algorithm was tested by performing a series of experiments during which we were interested in the values obtained for *byte hit ratio*. A number of 12 test instances were used.

We compare the results obtained when using utility based cache replacement strategies (utility functions (6.2) and (6.3) whose coefficients are generated using GeCo with the ones computed when classic cache replacement algorithms (LRU and LFU) [Podlipnig&Böszörményi2003] were used.

When performing the measurements, we made the assumption that no segmentation of video objects is used (web like treatment of objects) and that once requested, an object is fully retrieved from an origin server (if it is not already cached) and it is viewed from start to end with no interruptions or cancellations. Also no bandwidth limitations nor transmission errors were considered. While we are aware that those assumptions are unrealistic, the obtained results make a good reference point for the ideal case.

The data used for the experiments has been generated using WebTraff [Markatchev&Williamson2002] a synthetic web traffic generator. The characteristics of the 12 traces used are presented in Table 6.1.

Table 6.1 Characteristics of artificial trace logs

Trace ID	Number of requests	Number of objects	One-Timers(% of total objects)	Zipf slope
1	1000	300	70	0.3
2	1000	300	30	0.3
3	1000	300	70	0.75
4	1000	300	30	0.75
5	5000	1500	70	0.3
6	5000	1500	30	0.3
7	5000	1500	70	0.75
8	5000	1500	30	0.75
9	10000	3000	70	0.3
10	10000	3000	30	0.3
11	10000	3000	70	0.75
12	10000	3000	30	0.75

Our intent was to measure *byte hit ratio* under both lightly skewed (Zipf $\alpha = 0.3$) and more severe skewed (Zipf $\alpha = 0.75$) object popularity distribution and with varying amount of one-timers (objects requested only once) (30% for traces 2, 4, 6, 8, 10 and 12 vs. 70% in traces 1, 3, 5, 7, 9 and 11). We have also varied the size of the cache from 1% to 10% of the overall size of the requested objects (the size of all objects requested multiple times was considered only once).

The optimum value for each of the 12 traces was computed using the formula:

$$1 - \frac{TotalSizeOfUniqueObjects}{TotalSizeOfTransferredData} \quad (5)$$

The genetic algorithm (GeCo) we have used had the following setup: population size 100, number of generations 10, crossover probability 0.7 and mutation probability 0.6.

We present the results in terms of *byte hit rate* in Figure 6.1 and Figure 6.2 obtained for traces 11 and 12 with the coefficients (obtained using the GeCo algorithm). When studying the results it can be seen that using utility (6.2) yields much better results than when using utility (6.3) (Figure 6.1). In cases when the object popularity distribution is more severe skewed (Zipf α = 0.75) utility (6.2) clearly generates better results than LRU while in case the object popularity distribution is lightly skewed (Zipf α = 0.3) utility (6.2) provides comparable results with LRU (but still slightly better) (Figure 6.2).

Also it is interesting to notice that for large traces the increases in terms of *byte hit ratio* are negligible, for cache size greater than 7%. This means that we do not need extremely large cache sizes in order to obtain good *byte hit ratio* values which translates in reduced amount of used external bandwidth.

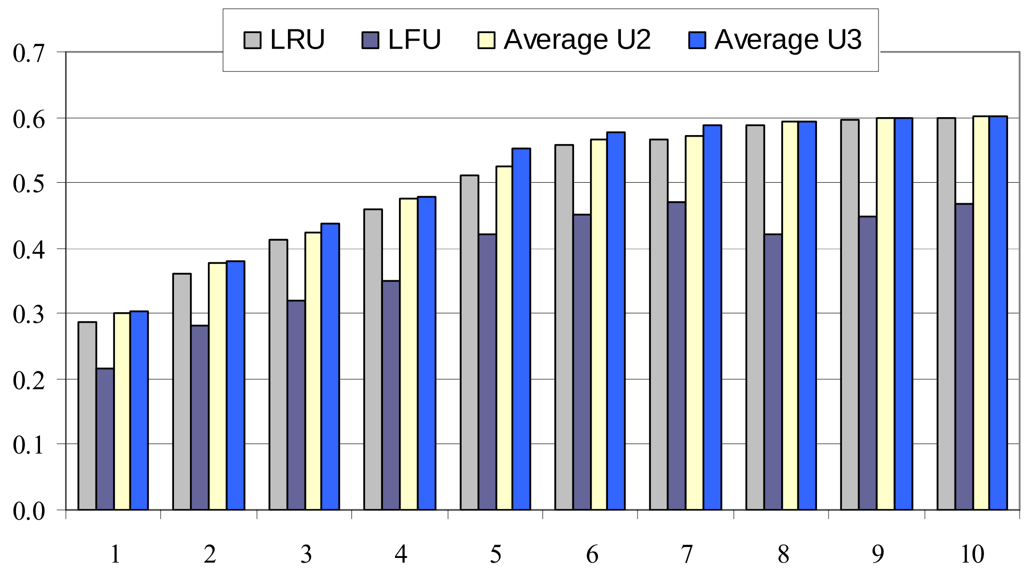


Figure 6.1 Byte hit ratio values for trace 11

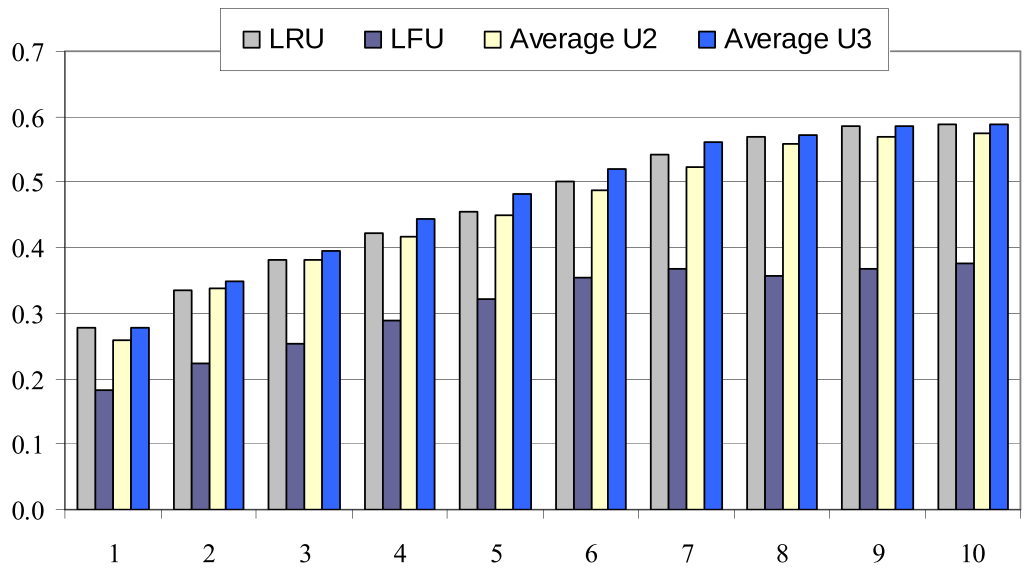


Figure 6.2 Byte hit ratio values for trace 12

Table 6.2 summarizes by exactly how much is utility (6.2) better than LRU when *byte hit rate* best and average results are considered. The average is about 1.7% which can be significant if pricing models are to be considered.

Table 6.2 Mean BHR difference between situations when using utility

Trace ID	Mean Best	Mean Average
1	1.0287%	0.6782%
2	1.1965%	0.9833%
3	4.0188%	3.7725%
4	0.3896%	0.1306%
5	1.1521%	0.6752%
6	0.3955%	0.0694%
7	2.4681%	2.1730%
8	0.8924%	0.6412%
9	1.0731%	0.8707%
10	0.7187%	0.4404%
11	1.8760%	1.7511%
12	1.3319%	1.1791%
Average	1.3784%	1.1137%

6.5. Conclusions and further research

We have proposed two new ways (equation (6.2) and (6.3)) of defining the utility of the objects stored inside a video proxy-cache that provide better balancing between considered characteristics (*size*, *timeLastAccess*, *hitCount*) for each object. We have also used a genetic algorithm (GeCo) for determining the coefficients that appear in those two definitions with the aim of maximizing the *byte hit rate*. The obtained results in terms of *byte hit rate* when the GeCo algorithm and the utility function (6.2) were used are similar or even better than those obtained for LRU.

We intend to use an approach similar to the one presented in GeCo in order to determine the coefficients that regulate the dynamics of the system proposed in [Cobârzan2005] and [Cobârzan&Böszörményi2007] namely the addition of new proxy-caching nodes to the federate cache, respectively their elimination when they are no longer needed. Considering segmentation strategies for the cached video objects while using the GeCo algorithm also needs further investigation.

Conclusions and further research

The aim of this thesis was to investigate the use of genetic algorithms in solving different classes of scheduling problems. Because the problems addressed are NP-hard the alternative of using genetic algorithms has been proven to be a good one.

In this respect basic elements of a scheduling problem (tasks, resources and objective functions), some aspects related to these elements, a classification scheme of scheduling problem and the complexity of these problems was presented first. Next an overview of the basic elements of a genetic algorithm (representation, fitness function, selection, crossover, mutation and parameters) was conducted.

Using the aforementioned background three scheduling problems were addressed:

- A uniform parallel machines scheduling problem was solved using a new genetic algorithm for the uniform parallel machines scheduling problem. The proposed algorithm obtained better results than other algorithms.
- A permutation flow shop scheduling problem was solved using a new hybrid genetic algorithm. The novelty of the proposed algorithm consisted in using a combination of a random initialization procedure and an initialization procedure based on NEH construction heuristic, in defining a new crossover operator and in using a mutation operator defined as a combination between an operator based on NEH construction heuristic shift mutation. The results obtained by the proposed algorithm are comparable with the results obtained by other algorithms.
- Video-proxy cache scheduling problem was addressed. For this problem two new formulas for utility of the objects stored inside a video proxy-cache were defined. A new genetic algorithm was used for determining the coefficients that appear in these two definitions with the aim of maximizing the byte hit rate. The results obtained by the proposed algorithm with one of utility formula are similar or even better than those obtained by other metric.

I intend to focus my future research on the following convergent direction:

- project management scheduling in order to investigate the field of tasks and resources allocation in projects that seems to be driving force for economic growth
- multi-objective optimization in order to consider more than just one objective function as a goal for optimization
- optimization in dynamic environments in order to simulate/capture the changes that appear during the lifecycle of a project
- stochastic scheduling in order to better simulate real life situations

References

[Abraham et. al. 2000] Abraham A., Buyya R. and Nath B., Nature's Heuristics for Scheduling Jobs in Computational Grids, in *Proceedings of 8th IEEE International Conference on Advanced Computing and Communications*, Tata McGraw-Hill Publishing Co. Ltd, New Delhi, pp. 45-52, 2000.

[Abraham et. al. 2006] Abraham A, Liu H, Zhang W, Chang TG, Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm, *Proceedings of 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems*, England, pp. 500-507, 2006.

[Abraham et. al. 2008] Abraham A, Liu H., Grosan C., and Xhafa F., *Nature Inspired Metaheuristics for Grid Scheduling: Single and Multiobjective Optimization Approaches, Metaheuristics for Scheduling: Distributed Computing Environments*, Studies in Computational Intelligence, Springer Verlag, Germany, pp. 247-272, 2008.

[Acharya2002] S. Acharya and B. Smith. Middleman: A video caching proxy server. In *Proceedings of the 10th International Workshop on Network and Operating System Support forDigital Audio and Video*, 2002.

[Ashlock2006] Ashlock, D., *Evolutionary Computation for Modeling and Optimization*, Springer, 2006

[Back et. al. 2000] Back T., D.B. Fogel, and Z. Michalewicz (Eds), *Evolutionary Computation: Basic Algorithms and Operators*, Vol. 1 and Vol. 2, Institute of Physics Publishing, Philadelphia, PA, 2000.

[Baeck2000] Baeck, T., Fogel, D., Michalewicz. (eds)., *Evolutionary Computation*, vol. 1 and 2, Institute of Physics Publishing, 2000.

[Balafoutis et. al. 2002] E. Balafoutis, A. Panagakis, N. Laoutaris, and I. Stavrakakis. The impact of replacement granularity on video caching. In *IFIP Networking 2002*, volume 2345 of *Lecture Notes in Computer Science*. Springer, 2002.

[Blazewicz et. al. 2007] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G. and Weglarz, J., *Handbook on scheduling. From Theory to Applications*, Springer, 2007.

[Blazewicz1983] Blazewicz J., Lenstra, J.K., A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics* 5, p. 11-24, 1883

[Brubeck&Brubeck1996] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed vod system. *IEEE MultiMedia*, 3(3):37–47, 1996.

[Brucker1999] Brucker, P., Drexl, A., Moehring, R., Neumann, K., Pesch, E., Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods, *European Journal of Operational Research*, no. 112, pp. 3-41, 1999.

[Brucker2007] Brucker, P., *Scheduling Algorithms*, Springer, 2007

[Budiu1999] Budiu, M., 1999, http://www.cs.cmu.edu/~mihaib/articole/

[Buyya et. al. 2000] Buyya R, Abramson D, Giddy J, Grid Resource Management, Scheduling, and Computational Economy, *International Workshop on Global and Cluster Computing,* Japan, 2000.

[Chen et. al. 1995] Chen, C.-L., Vempati, V. S., and Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. European Journal of Operational Research, 80(2):389–396.

[Cobârzan&Böszörményi2007] C. Cobârzan and L. Böszörményi. Further developments of a dynamic distributed video proxy-cache system. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007)*, pages 349–357. IEEE Computer Society, 2007.

[Cobârzan2005] C. Cobârzan. Dynamic proxy-cache multiplication inside LANs. In *Euro-Par 2005*, volume 3648 of *Lecture Notes in Computer Science*, pages 890–900. Springer, 2005.

[Cobârzan&Mihăilă2008] Cobârzan C., **Mihăilă C.**, A Genetic Algorithm for Utility Based Video Proxy-Caching, in *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 231-238, Timişoara, România, 2008

[Conway2003] R. Conway, W. Maxwell and L. Miller. *Theory of scheduling*. Dover Publications Inc., reprint edition, 2003.

[Cook1971] Cook, S., The Complexity of Theorem Proving Procedures, *in Proceedings of the third annual ACM symposium on Theory of computing*, pp.151–158, 1971.

[Dannenbring1977] Dannenbring, D. G. An evaluation of flow shop sequencing heuristics. Management Science, 23, 11 (1977), 1174--1182.

[Demeulemeester2002] Demeulemeester, E.L., Herroelen, W.S., *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers, Dordrecht, 2002.

[Drozdowski1996] Drozdowski M., *Selected Problems of Scheduling Tasks in Multiprocessor Computer Systems*, Poznan University of Technology Press, Poznan, 1996.

[Dumitrescu2000] Dumitrescu, D., Lazzerini, B., Jain, L., Dumitrescu, A., *Evolutionary Computation.* CRC Press, Boca Raton, FL, 2000.

[Dumitrescu et.al.2002a] Dumitrescu, D., Iantovics, B., **Florea, C.**, Multi-Agent Systems: a new allocation protocol and evolutionary search for equilibrium; , in *Proceedings of the Symposium "Zilele academice clujene" - Computer Science Section*, 119-133, Cluj-Napoca, Romania, 2002.

[Dumitrescu et.al.2002b]Dumitrescu, D., **Florea, C.**, Patranjan; P., Evolutionary Reorganization in MAS; , in *Proceedings of the European Conference on Information Technology* (ECIT02), 1-5, Iasi, Romania, 2002.

[Dumitrescu et.al.2002c]Dumitrescu, D., **Florea, C.**, Patranjan; P., A New Evolutionary Model for Multi Agent Systems, in *Proceedings of* the *4th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC02), 137-143, Timisoara, Romania, 2002.

[Eiben2003] Eiben, A.E., Smith, J.E., *Introduction to Evolutionary Computing*, Springer, 2003.

[Fabmi et. al. 2001] H. Fabmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, and L. Hsu. Proxy servers for scalable interactive video support. *Computer*, 34(9):54–60, 2001.

[Florea&Dumitrescu2003] **Florea, C**., Dumitrescu, D., Negotiation in Multiagent Systems; in *Proceedings of the Conference on Applied and Industrial Mathematics* (CAIM03), Oradea, Romania, 2003.

[Fogel1966] Fogel, L.J., Owens, A.J., Walsh, M.J., *Artificial Intelligence through Simulated Evolution*, John Wiley, 1966.

[Garey&Johnson1979] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman, New York, 1979.

[Garey1979] Garey, M.R., Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, WH Freeman&Co, San Francisco, 1979.

[Goldberg1989] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning,* Kluwer Academic Publishers, Boston, MA, 1989.

[Grabowski&Wodecki2004] Grabowski, J. and Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research, 31(11):1891–1909.

[Graham1979] Graham, R.L., Lawler, E.L., Lenstra, J.K., A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, *Annals of Discrete Mathematics* 5, p. 287-326, 1979

[Gro□an et.al.2003] Groşan, C., Oltean, M., **Florea, C.**, NP-complete problems using Evolutionary Algorithms, *Lucrarile Seminarului de Didactica Matematicii al Universitatii Babes-Bolyai*, Vadu-Crisului, Romania, 2003.

[Grosan et. al. 2007] Grosan, C., Abraham, A., and Helvik, B., Multiobjective Evolutionary Algorithms for Scheduling Jobs on Computational Grids, *IADIS International Conference, Applied Computing 2007*, pp. 459-463, 2007.

[Guangchang et. al. 2006] Guangchang Ye, Ruonan Rao, Minglu Li, A Multiobjective Resources Scheduling Approach Based on Genetic Algorithms in Grid Environment, Fifth International Conference on Grid and Cooperative Computing Workshops, pp. 504-509, 2006.

[Holland1975] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[HsiuMa&Du2000] W. Hsiu Ma and D. H.-C. Du. Reducing bandwidth requirement for delivering video over wide area networks with proxy server. In *IEEE International Conference onMultimedia and Expo (II)*, pages 991–994. IEEE Computer Society, 2000.

[Johnson1954] Johnson, S. M. Optimal two-and three-stage production schedules. Naval Research Logistics Quarterly, 1 (1954), 61--68.

[Karp1972] Karp, R.M., Reducibility Among Combinatorial Problems, *Complexity of Computer Computations*, pp. 85-103, Plenum Press, 1972

[Kumar et. al. 2004] Kumar , K.P., Agarwal , A., and Krishnan, R., Fuzzy based resource management framework for high throughput computing, in *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, 555-562, 2004.

[Leung2000] Leung, J.Y-T. (ed.), *Handbook of Scheduling. Algorithms, Models and Performance Analysis,* Chapman & Hall/CRC Press, Boca Raton, 2000.

[Leung2004] Leung, J.Y-T, Anderson, J.H., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman and Hall / CRC, Boca Raton, Florida, 2004.

[Liu1995] Liu, Z., Sanlaville, E., Preemptive scheduling with variable profile, precedence constraints and due dates, Discrete Applied Mathematics 58, p. 253-280, 1995

[Markatchev&Williamson2002] N. Markatchev and C. Williamson. Webtraff: A gui for web proxy cache workload modeling and analysis. In *MASCOTS'02: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02).* IEEE Computer Society, 2002.

[Mihăilă&Cobârzan2008] **Mihăilă C.**, Cobârzan C., Evolutionary approach for multimedia caching, in *IEEE Proceedings of the Evolutionary Techniques in Data Processing Workshop,* International Conference on Database and Expert Systems Application (DEXA), 531-536, Torino, Italia, 2008

[Mihăilă&Dumitrescu] **Mihăilă, C.**, Dumitrescu, D., Quantum Computing and Multiagent Systems, in *Proceedings of the Symposium "Colocviul Academic Clujean de Informatica",* Cluj-Napoca, Romania, 2005.

[Mihăilă&Mihăilă2008a] **Mihăilă C.**, Mihăilă A., An Evolutionary Algorithm for Uniform Parallel Machines Scheduling, in *IEEE Proceedings of the European Modelling Symposium*, 76-80, Liverpool, United Kingdom, 2008.

[Mihăilă&Mihăilă2008b] Mihăilă A., **Mihăilă C.**, Uniform Parallel Machines Scheduling using an Evolutionary Algorithm, in *IEEE Proceedings of the International Workshop on Evolutionary Multiobjective Optimization Design and Applications*, International

Conference on Intelligent Systems Design and Applications (ISDA), 401-406, Kaohsiung, Taiwan, 2008

[Mihăilă et.al.2008a] Mihăilă A., Mihiş A., **Mihăilă C.,** Genetic Algorithm for Logical Topic Text Segmentation, in *IEEE Proceedings of the International Conference on Digital Information Management*, 500-505, London, United Kingdom, 2008

[Mihăilă et.al.2008b] **Mihăilă C**., Nițchi, I.Ş., R., Mihăilă, A., Coroş R., A genetic algorithm for permutation flow shop scheduling problem, in Annals of Tiberiu Popoviciu Seminar of Functional Equation, Approximation and Convexity, p. 241-250, Cluj-Napoca, România, 2008

[Mihi□ et.al.2006] Mihiş, A., Crețu, C., **Mihăilă, C.**, Şerban, C., Code Simplification using Boolean Functions Simplification, in *Proceedings of the International Conference of Mathematics & Informatics*, Supplement of "Studii si cercetari stiintifice. Seria Matematică", no.16, University of Bacau, 493-502, Bacău, Romania, 2006.

[Montana2007]Montana, D., Hussain, T., Vidaver, G., A Genetic-Algorithm-Based Reconfigurable Scheduler, *Evolutionary Scheduling,* Springer, 2007.

[Murata et. al. 1996] Murata, T., Ishibuchi, H., and Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. Computers & Industrial Engineering, 30(4):1061–1071.

[Nawaz et. al. 1983] Nawaz, M., Enscore Jr., E. E., and Ham, I. A heuristic algorithm for the m-machine, n-job flow shop sequencing problem. OMEGA, 11, 1 (1983), 91--95.

[Nițchi et.al.2007a] Nițchi, I.Ş., Mihăilă, A., **Mihăilă, C.**, About Project Management Planning Optimization using Genetic Algorithms, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 79-82 ,Cluj-Napoca, România, 2007.

[Nițchi et.al.2007b] Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A., **Mihăilă, C.**, About the Logical Model for Intelligent Agents, in Proceedings of the International Conference on Knowledge Engineering Principles and Technologies, Special issue of *Studia Universitatis Babes-Bolyai Informatica Series*, 83-90, Cluj-Napoca, România, 2007.

[Nițchi et.al.2007c] Nițchi, I.Ş., Avram-Nițchi, R., Mihăilă, A**.**, **Mihăilă, C.**, On the collaborative systems for e-business, in *Proceedings of the International Conference on Competitiveness and European Integration*, 266-272, Cluj-Napoca, România, 2007.

[Nowicki&Smutnicki1996] Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flowshop problem. European Journal of Operational Research, 91(1):160–175.

[Oltean et.al.2009] Oltean M., Groşan C., Dioşan L., **Mihăilă C.,** *Genetic Programming with Linear Representation a Survey,* International Journal on Artificial Intelligence Tools, 197-238, 2009

[Osman&Potts1989] Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. OMEGA, The International Journal of Management Science, 17(6):551–557.

[Park et. al. 2001] S.-H. Park, E.-J. Lim, and K.-D. Chung. Popularity-based partial caching for vod systems using a proxy server. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01)*. IEEE Computer Society, 2001.

[Podlipnig&Böszörményi2002] S. Podlipnig and L. Böszörményi. Replacement strategies for quality based video caching. In *IEEE International Conference on Multimedia and Expo (ICME), Vol. 2*, pages 49–53. IEEE Computer Society, 2002.

[Podlipnig&Böszörményi2003] S. Podlipnig and L. Böszörményi. A survey of web cache replacement strategies. *ACMComput. Surv.*, 35(4):374–398, 2003.

[Rajendran&Ziegler2004] Rajendran, C., and Ziegler, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research, 155, 2 (2004), 426--438.

[Rechenberg1973] Rechenberg, I., *Evolution Strategy* , Frommann-Holzboog, Stuttgart, pp.147-159, 1973.

[Reeves&Yamada1998] Reeves, C. and Yamada, T. (1998). Genetic algorithms, path relinking, and the flowshop sequencing problem. Evolutionary Computation, 6(1):45–60.

[Rejaie&Kangasharju2001] R. Rejaie and J. Kangasharju. Mocha: A quality adaptive multimedia proxy cache for internet streaming. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 3–10. ACM Press, 2001.

[Ritchie&Levine2003] Ritchie, G. and Levine, J., A fast, effective local search for scheduling independent jobs in heterogeneous computing environments, Technical report, Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh, 2003.

[Ritchie&Levine2004] Ritchie, G. and Levine, J., A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in *23rd Workshop of the UK Planning and Scheduling Special Interest Group*, 2004.

[Ruiz et. al. 2004] Ruiz, R., Maroto, C., and Alcaraz, J. (2004). Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA, the International Journal of Management Science.

[Ruiz&Maroto2004] Ruiz, R. and Maroto, C. (2004). A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research. In press.

[Ruiz&Stützle2007] Ruiz, R., and Stützle, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177 (2007), 2033--2049.

[Sasabe et. al. 2001] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara. Proxy caching mechanisms with video quality adjustment. In *Proceedings of SPIE International Symposium on The Convergence of Information Technologies and Communications*, pages 276–284, 2001.

[Schmidt1984] Schmidt, G., Scheduling on semi-identical processors, Zeitschrift für Operations Research. A28, p. 153-162, 1984

[Sen et. al. 1999] S. Sen, J. Rexford, and D. F. Towsley. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM*, pages 1310–1319. IEEE Computer Society, 1999.

[Smith 2005] Smith, S.F., *Is Scheduling a Solved Problem?*, Multidisciplinary Scheduling: Theory and Applications, Springer, pp. 3-17, 2005.

[Suliman2000] Suliman, S. (2000). A two-phase heuristic approach to the permutation flow-shop scheduling problem. International Journal of Production Economics, 64:143–152.

[Taillard1990] Taillard, E. Some efficient heuristic methods for the flowshop sequencing problems. European Journal of Operational Research, 47 (1990), 65--74.

[Taillard1993] Taillard, E. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, 64(2):278–285.

[Taillard2004] Taillard, E. (2004). Summary of best known lower and upper bounds of Taillard's instances. http://mistic.heig-vd.ch/taillard/.

[Tompkins2003] Tompkins, M. F., *Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations*, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[Veltman1993] Veltman, B., Multiprocessor Scheduling with Communication Delays, Ph.D Thesis, CWI-Amstrerdam, 1993.

[Weawer2006] Weawer Patrick, A brief history of scheduling, myPrimavera06, Canberra, 2006, http://www.pmforum.org/library/papers/2006/A_Brief_History_of_Scheduling.pdf

[Wu et. al. 2001] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 36–44. ACM Press, 2001.

[Yang& Towsley2002] S. S. Yang Guo and D. Towsley. Prefix caching assisted periodic broadcast for streaming popular videos. In *Proceedings of ICC (International Conference on Communications)*, pages 2607 – 2612. IEEE Computer Society, 2002.

[Yarkhan&Dongarra2002] Yarkhan, A. and Dongarra, J., Experiments with scheduling using simulated annealing in a grid environment, in *3rd International Workshop on Grid Computing* (GRID2002), 232-242, 2002.

[Zomaya&The2001] Zomaya, A.Y. and The, Y.H., Observations on using genetic algorithms for dynamic load-balancing, *IEEE Transactions On Parallel and Distributed Systems*, 12(9):899-911, 2001.