International Handbooks on Information Systems

Christoph Schwindt
Jürgen Zimmermann *Editors*

Handbook on Project Management and Scheduling Vol. 2

Springer

Chapter 31
The Basic Multi-Project Scheduling Problem

José Fernando Gonçalves, Jorge José de Magalhães Mendes, and Mauricio G.C. Resende

Abstract In this chapter the Basic Multi-Project Scheduling Problem (BMPSP) is described, an overview of the literature on multi-project scheduling is provided, and a solution approach based on a biased random-key genetic algorithm (BRKGA) is presented. The BMPSP consists in finding a schedule for all the activities belonging to all the projects taking into account the precedence constraints and the availability of resources, while minimizing some measure of performance. The representation of the problem is based on random keys. The BRKGA generates priorities, delay times, and release dates, which are used by a heuristic decoder procedure to construct parameterized active schedules. The performance of the proposed approach is validated on a set of randomly generated problems.

Keywords Genetic algorithm • Meta-heuristics • Multi-project scheduling • Random keys

31.1 Introduction

Managing multiple projects is a complex decision-making process, where a number of projects must share concurrently a set of limited resources. Examples of multi-project environments are new product development, multi-product manufacturing, infrastructure constructions, and maintenance of systems.

J.F. Gonçalves (✉)
LIAAD, INESC TEC, Faculdade de Economia da Universidade do Porto, Porto, Portugal
e-mail: jfgoncal@fep.up.pt

J.J.M. de Mendes
Instituto Superior de Engenharia do Porto, Depto. de Engenharia Civil, Porto, Portugal
e-mail: jjm@isep.ipp.pt

M.G.C. Resende
Algorithms and Optimization Research Department, AT&T Labs Research, Florham Park, NJ, USA
e-mail: mgcr@research.att.com

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 2*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05915-0_1

The basic multi-project scheduling problem (BMPSP) can be considered an extension of the well-known resource constrained project scheduling problem (RCPSP) where two or more projects which require the same scarce resources are scheduled simultaneously.

There are two main distinguished research fields in multi-project scheduling—the static and the dynamic project environment (Dumond and Mabert 1988). In this chapter we assume a closed project portfolio, which does not change over time. The BMPSP in a static environment has been studied, amongst others, by Fendley (1968), Pritsker et al. (1969), Kurtulus and Davis (1982), Kurtulus and Narula (1985), Lawrence and Morton (1993), Lova et al. (2000), Lova and Tormos (2001, 2002), Gonçalves et al. (2008), Krüger and Scholl (2010), Browning and Yassine (2010), Kumanam and Raja (2011), and Cai and Li (2012).

The existing solution methods apply either a single- or a multi-project approach. The single-project approach is equivalent to the RCPSP, since it merges all projects of the multi-project into an artificial super-project with dummy start and end activities. The multi-project approach keeps the projects separate. The approach considered in this chapter uses a single-project approach.

Scheduling involves the allocation of the given resources to projects to determine the start and completion times of a set of detailed activities. There may be multiple projects contending for limited resources, which makes the solution process more complex. The allocation of scarce resources then becomes a major objective of the problem and several compromises have to be made to solve the problem to the desired level of near-optimality.

In this chapter, we present a biased random-key genetic algorithm (BRKGA) approach to solve the BMPSP. The remainder of the chapter is organized as follows. Section 31.2 describes the problem and presents a conceptual model and Sect. 31.3 reviews the literature. Section 31.4 describes the approach used to solve the BMPSP. Section 31.5 describes the parameterized schedule-generation procedure and Sect. 31.6 reports on some computational experiments. Concluding remarks are made in Sect. 31.7.

31.2 Problem Description

The BMPSP consists of a set Q of projects, where each project $q \in Q$ is composed of activities $j \in V_q$, where activities α_q and ω_q are dummies and represent, respectively, the initial and final activities of project q. Let V be the set of all activities and let $\mathscr{R} = \{1, \ldots, K\}$ represent the set of renewable resources. While being processed, activity $j \in V$ requires r_{jk} units of resource $k \in \mathscr{R}$ during each time instant of its non-preemptable duration p_j. Resource $k \in \mathscr{R}$ has a limited availability of R_k at any point in time. Parameters p_j, r_{jk}, and R_k are assumed to be non-negative and deterministic. The activities are interrelated by the following two kinds of constraints:

- *Precedence constraints*, which force each activity $i \in V$ to be scheduled after all predecessor activities $j \in Pred(i)$ are completed;
- *Resource constraints,* which assure that the processing of the activities is subject to the availability of resources with limited capacities.

For the start and end activities of each project q, we have, for all $q \in Q$, that

$$p_{\alpha_q} = p_{w_q} = 0 \quad \text{and} \quad r_{\alpha_q k} = r_{w_q k} = 0 \quad (k \in \mathscr{R})$$

Activities 0 and $n+1$ are dummy activities, have no duration, and correspond to the start and end of all projects.

The BMPSP consists in finding a schedule for all the activities taking into account precedence constraints and the availability of resources, while minimizing some measure of performance. Let C_j represent the finish time of activity $j \in V$. A schedule can be represented by a vector of finish times $(C_1, \ldots, C_{n+1})$. Let $\mathscr{A}(t)$ be the set of activities being processed at time t. The conceptual model of the BMPSP can be described as

$$\text{Min.} \quad \textit{Measure of Performance}\,(C_1, \ldots, C_n) \tag{31.1}$$

s.t.

$$C_i \leq C_j - p_j \quad (j \in V;\ i \in Pred(j)) \tag{31.2}$$

$$\sum_{j \in \mathscr{A}(t)} r_{jk} \leq R_k \quad (k \in \mathscr{R};\ t \geq 0) \tag{31.3}$$

$$C_j \geq 0 \quad (j \in V) \tag{31.4}$$

According to objective (31.1) we seek to minimize some performance measure. Constraints (31.2) impose the precedence relations between activities, and constraints (31.3) limit the resource usage imposed by the activities being processed at time t to the available capacity. Finally, constraints (31.4) force the finish times to be non-negative.

A variety of measures of performance have been used for the BMPSP. Minimization of project duration has been used widely (Baker 1974). Other measures of performance include: minimization of total project delay, lateness, or tardiness (Kurtulus and Davis 1982), minimization of average project delay (Lova and Tormos 2001), minimization of total lateness or lateness penalty (Kurtulus 1985), minimization of the overall project cost (Talbot 1982), minimization the total cost of delay (Kurtulus and Narula 1985), and maximization of the resource leveling (Woodworth and Willie 1975). In this chapter, we seek to minimize a measure of performance which involves the due date (tardiness), starting time (earliness), and work in process (flow time) of each project (Gonçalves et al. 2008). This performance measure simultaneously incorporates three criteria: tardiness, earliness, and flow time and is described below. The following notation will be used:

$\hat{p}_q$: Target duration for project q.
d_q: Due date for project q.
C_q: Conclusion date for project q in the generated schedule.
S_q: Start date for project q in the generated schedule.
T_q: Tardiness of project $q = \max\{C_q - d_q\ ,\ 0\}$.
E_q: Earliness of project $q = \max\{d_q - C_q\ ,\ 0\}$.
FD_q: Flow time deviation for project $q = \max\{C_q\ -\ S_q\ -\ \hat{p}_q,\ 0\}$.
LB_0^q: Critical path length of project q.

the performance measure is defined as

$$w^T \sum_q T_q^3 \quad + \quad w^E \sum_q E_q^2 \quad + \quad w^{FD} \sum_q FD_q^2 \tag{31.5}$$

where w^T, w^F, and w^{FD} are parameters defined by the decision maker. Note that the tardiness has an exponent equal to 3 because in the real-world it is considered more important than the earliness or the flow-time (which have an exponent equal to 2). To overcome the problem of not knowing the target duration of a project in a real-world situation, we replace

$$w^T \sum_q FD_q^2 \quad \text{by} \quad w^T \sum_q \frac{\left(C_q - S_q\right)^2}{LB_0^q}$$

In the above model, the constraints for the resources are expressed by condition (31.3). However, there are others types of constraints related with the start of a project which cannot be modeled by that condition. To be able to model this kind of constraint, we add

$$C_{\alpha_q} \geq ES_q \qquad (q \in Q)$$

to the model, where ES_q represents earliest release date for project q. These constraints are enforced in the model implicitly by assigning to the initial activity of each project a duration $ES_q \geq \overline{ES}_q$, i.e.,

$$p_{\alpha_q} = ES_q \geq \overline{ES}_q \qquad (q \in Q)$$

31.3 Literature Review

The BMPSP is a generalization of the RCPSP. Blazewicz et al. (1983) show that the RCPSP, as a generalization of the classical job shop scheduling problem, belongs to the class of $\mathcal{NP}$-hard optimization problems. Therefore the BMPSP, as a generalization of the RCPSP, is also $\mathcal{NP}$-hard.

Exact methods to solve the BMPSP are proposed in the literature. The pioneering work of multi-project scheduling by Pritsker et al. (1969) proposed a zero-one programming approach. Mohanthy and Siddiq (1989) studied the problem of assigning due dates to the projects in a multi-project environment. Drexl (1991) considered a non-preemptive variant of the resource-constrained assignment problem using a hybrid branch-and-bound/dynamic programming algorithm with a Monte Carlo-type upper bounding heuristic. Deckro et al. (1991) formulated the BMPSP as a block angular general integer programming model and employed a decomposition approach to solve large problems. Vercellis (1994) describes a Lagrangian decomposition technique for solving multi-project planning problems with resource constraints and alternative modes of performing each activity in the projects.

Several approaches to the BMPSP using heuristic methods have been proposed in the literature. For example, Fendley (1968) used multi-projects with three and five projects and considered three efficiency measurements in the computational analysis. Kurtulus and Davis (1982) designed multi-project instances whose projects have between 34 and 63 activities and resource requirements for each activity between two and six units.

Kurtulus and Narula (1985) studied penalties due to project delay. Dumond and Mabert (1988) studied the problem of assigning due dates to the projects in a multi-project environment. Tsubakitani and Deckro (1990) proposed a heuristic for multi-project scheduling with resource constraints using the Kurtulus and Davis (1982) approach to select appropriate heuristic decision rules. Bock and Patterson (1990) designed a computational experiment based on the work of Dumond and Mabert (1988) with three factors. Lawrence and Morton (1993) studied the due date setting problem of scheduling multiple resource-constrained projects with the objective of minimizing weighted tardiness costs. Shankar and Nagi (1996) proposed a two-level hierarchical approach consisting of the planning and scheduling stages.

Özdamar et al. (1998) examined different dispatching rules for the tardiness and the net present value objective embedded in a multi-pass heuristic. Ash (1999) proposed a deterministic simulation scheme using available project data to choose an activity scheduling heuristic which not only allows for the establishment of good project schedules, but determines a priori which resources will be assigned to specific project activities.

Lova et al. (2000) developed a multi-criteria heuristic that, lexicographically, improves two criteria: a temporal criterion (mean project delay in relation to the unconstrained critical path duration or multi-project duration increase) and a non-temporal criterion (project splitting, in-process inventory, resource leveling, or idle resources) that can be chosen by the user.

Mendes (2003) presents a genetic algorithm that uses a random-key representation and a modified parallel schedule-generation scheme (SGS).

31.4 Biased Random-Key Genetic Algorithm

We begin this section with an overview of the proposed solution process. This is followed by a discussion of the biased random-key genetic algorithm, including detailed descriptions of the solution encoding and decoding, evolutionary process, and fitness function.

31.4.1 Overview

Considering the difficulty to solve real-world problems with exact methods, a new solution approach is developed that combines a genetic algorithm with a schedule-generation scheme (SGS) that creates parameterized active schedules. The SGS constructs a schedule based on the priorities and delay times of the activities, and the release dates of the projects.

The role of the genetic algorithm (GA) is to evolve the encoded solutions, or *chromosomes*, which encode the vectors of priorities (Π) and delays (Δ) of the activities and the vector of project release dates (ES). For each chromosome, the following phases are applied to decode the chromosome:

1. *Decoding of the priorities*. This phase transforms a part of the chromosome supplied by the genetic algorithm into the vector of activity priorities (Π).
2. *Decoding of the delay times*. This phase transforms a part of the chromosome supplied by the genetic algorithm into the vector of activity delays (Δ).
3. *Decoding of the release dates*. This phase transforms a part of the chromosome supplied by the genetic algorithm into the vector of project release dates (ES).
4. *Schedule generation*. This phase makes use of Π, Δ, and ES, generated in the previous phases, and constructs parameterized active schedules.
5. *Fitness evaluation*: This phase computes the fitness of the solution (or measure of quality of the schedule) according to Eq. (31.5).

Figure 31.1 illustrates the sequence of steps applied to each chromosome generated by the BRKGA.

The remainder of this section details the genetic algorithm, the decoding procedure, and the SGS

31.4.2 Biased Random-Key Genetic Algorithm

Genetic algorithms with random keys, or *random-key genetic algorithms* (RKGA), for solving problems like sequencing, whose solutions can be encoded as permutation vectors, were introduced in Bean (1994). In an RKGA, chromosomes are represented as vectors of randomly generated real numbers in the interval $[0, 1]$.

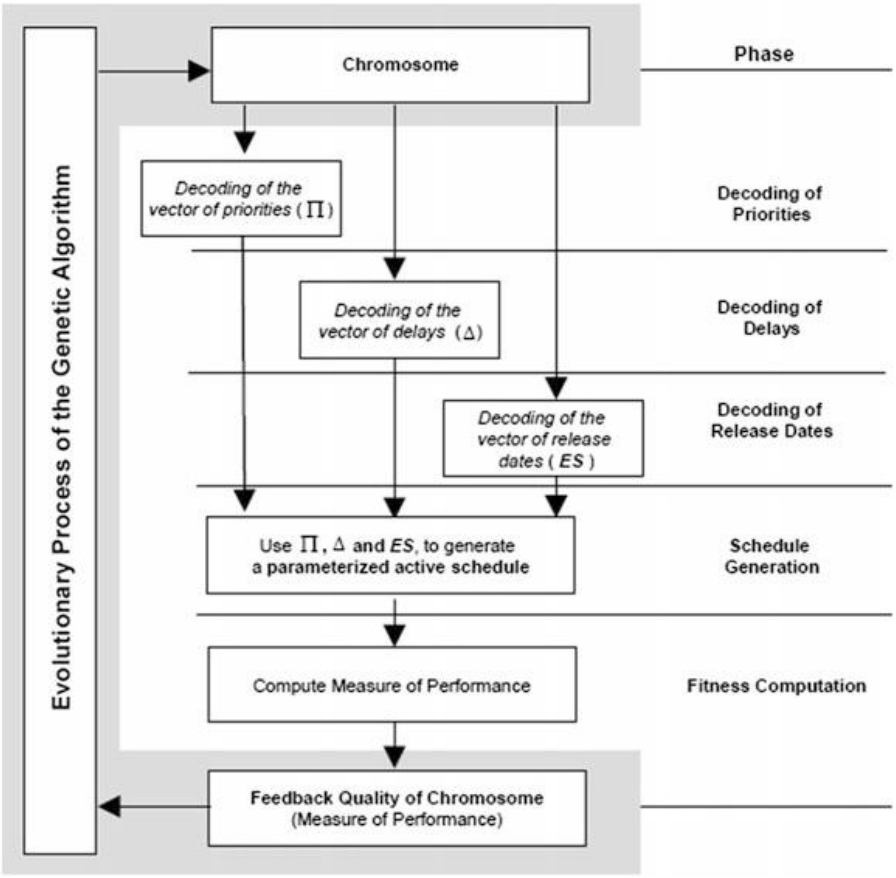

Fig. 31.1 Architecture of the algorithm

The *decoder*, a deterministic algorithm, takes as input a chromosome and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

Random key GAs are particularly attractive for sequencing problems and/or when the chromosomes have several parts (see, for example, Gonçalves and Almeida 2002; Gonçalves and Resende 2004; Gonçalves and Sousa 2011). Unlike traditional GAs, which need to use special repair procedures to handle permutations or sequences, RKGAs move all the feasibility issues into the objective evaluation procedure and guarantee that all offspring formed by crossover correspond to feasible solutions. When the chromosomes have several parts, traditional GAs need to use different genetic operators for each part. However, since RKGAs use parametrized uniform crossovers (instead of the traditional one-point or two-point crossover), they do not need to have different genetic operators for each part.

A *RKGA* evolves a *population* of random-key vectors over a number of *generations* (iterations). The initial population is made up of σ_{pop}^{init} vectors of n_{key} random keys. Each component of the solution vector, or random key, is generated independently at random in the real interval $[0, 1]$. Next, the fitness of each individual is computed by the decoder in generation g, the population is partitioned

into two groups of individuals: a small group of $n_{elit} < \sigma_{pop}^{init}/2$ *elite* individuals, i.e., those with the best fitness values, and the remaining set of $\sigma_{pop}^{init} - n_{elit}$ *non-elite* individuals. To evolve the population of generation g, a new generation of individuals is produced. All elite individuals of the population of generation g are copied without modification to the population of generation $g + 1$. *RKGAs* implement mutation by introducing *mutants* into the population. A mutant is a vector of random keys generated in the same way in which an element of the initial population is generated. At each generation, a small number $n_{mut} < \sigma_{pop}^{init}/2$ of mutants is introduced into the population. With $n_{elit} + n_{mut}$ individuals accounted for in the population of generation $g + 1$, $\sigma_{pop}^{init} - n_{elit} - n_{mut}$ additional individuals need to be generated to complete the σ_{pop}^{init} individuals that make up population $g+1$. This is done by producing $\sigma_{pop}^{init} - n_{elit} - n_{mut}$ offspring solutions through the process of *mating* or *crossover*.

A *biased random-key genetic algorithm,* or *BRKGA* (Gonçalves and Resende 2011a), differs from a *RKGA* in the way parents are selected for mating. While in the *RKGA* of Bean (1994) both parents are selected at random from the entire current population, in a *BRKGA* each offspring is generated combining a parent selected at random from the elite partition in the current population and one selected at random from the rest of the population. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring in the same generation. As in *RKGAs*, *parameterized uniform crossover* (Spears and Dejong 1991) is used to implement mating in *BRKGAs*. Let π_{elit} be the probability that an offspring inherits the vector component of its elite parent. Recall that n_{key} denotes the number of components in the solution vector of an individual. For $l = 1, \ldots, n_{key}$, the l-th component $c(l)$ of the offspring vector c takes on the value of the l-th component $e(l)$ of the elite parent e with probability π_{elit} and the value of the l-th component $\bar{e}(l)$ of the non-elite parent $\bar{e}$ with probability $1 - \pi_{elit}$.

When the next population is complete, i.e., when it has σ_{pop}^{init} individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

A *BRKGA* searches the solution space of the combinatorial optimization problem indirectly by searching the continuous n_{key}-dimensional hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

To specify a biased random-key genetic algorithm, we simply need to specify how solutions are encoded and decoded and how their corresponding fitness values are computed. We specify our algorithm next by first showing how the resource-constrained multi-project scheduling solutions are encoded and then decoded and how their fitness evaluation is performed.

31.4.3 Chromosome Representation

A chromosome represents a solution to the problem and is encoded as a vector of random keys. In a direct representation, a chromosome represents a solution of the original problem, and is usually called *genotype*, while in an indirect representation it does not and special procedures are needed to derive a solution from it usually called *phenotype*.

In the present context, the direct use of schedules as chromosomes is too complicated to represent and manipulate. In particular, it is difficult to develop corresponding crossover and mutation operations. Instead, solutions are represented indirectly by parameters that are later used by a schedule generator to obtain a solution. To obtain the solution (phenotype) we use the parameterized active schedule generator described in Sect. 31.5. Each solution chromosome is made of $2n + m$ genes, where n is the number of activities and m is the number of projects:

$$Chromosome = (\underbrace{gene_1, \ldots, gene_n}_{\text{Priorities}}, \underbrace{gene_{n+1}, \ldots, gene_{2n}}_{\text{Delay Times}}, \underbrace{gene_{2n+1}, \ldots, gene_{2n+m}}_{\text{Release Dates}})$$

The first n genes are used to determine the priorities of each activity. The genes $n + 1$ to $2n$ are used to determine the delay time used at each of the n iterations of the scheduling procedure, which schedules one activity per iteration. The last m genes are used to determine the release dates of each of the m projects.

31.4.4 Decoding of the Activity Priorities

As mentioned in Sect. 31.4.3, the first n genes are used to obtain activity priorities. Activity priorities are values between 0 and 1. The higher the value, the higher the priority will be. Below, we present the decoding procedure for the activity priorities.

Let $TF_j = d_{q(j)} - l_j$, represent the slack of activity j where $d_{q(j)}$ is the due date of the project q to which activity j belongs and l_j is the length of the longest-length path from the beginning of activity j to the end of the project $q(j)$ to which activity j belongs. Furthermore, let TF^{max} be the maximum slack for all activities amongst all projects.

The priority of each activity j is given by an expression which produces priority values between 70 and 100 % of the normalized slack. The priority of each activity j is given by the following expression

$$\Pi_j = \frac{TF_j}{TF^{max}} \times \left(0.7 + 0.3 \times gene_j\right)$$

31.4.5 Decoding of the Delays

Genes $n+1$ to $2n$ are used to determine the delay times Δ_i, used by each scheduling iteration i. The delay time for each activity i is calculated by

$$\Delta_i = gene_i \times 1.5 \times p^{max}$$

where p^{max} is the maximum duration amongst all activity durations. The factor 1.5 was obtained after experimenting with values between 1.0 and 2.0 in increments of 0.1.

31.4.6 Decoding of the Release Dates

The last m genes of each the chromosome (genes $2n + 1$ to $2n + m$) are used to determine the release dates of each project $q \in Q$. The following decoding expression is used to obtain the release date of each project $q \in Q$:

$$ES_q = \overline{ES}_q + gene_{2n+q} \times \left(d_q - \overline{ES}_q\right)$$

Consequently, the duration of the initial activity of each project q is equal to

$$p_{\alpha_q} = ES_q \qquad (q \in Q)$$

31.5 Schedule-Generation Procedure

The set of active schedules is usually very large and contains many schedules with relatively large delay times, having therefore poor quality in terms of the performance measure. To reduce the solution space, parameterized active schedules, introduced by Gonçalves and Beirão (1999) and Gonçalves et al. (2005) are used. The basic idea of parameterized active schedules consists in controlling the delay time allowed for each activity to encounter. By controlling the maximum delay time allowed, one can reduce or increase the solution space. A maximum delay time equal to zero is equivalent to restricting the solution space to non-delay schedules and a maximum delay time equal to infinity is equivalent to allowing general active schedules.

The procedure used to construct parameterized active schedules is based on a schedule-generation scheme that proceeds by time-increments. For each iteration μ, there is a scheduling time t_μ. All activities which are active at t_μ form the active set, i.e.,

$$\mathscr{A}_\mu = \left\{ j \in V \mid C_j - d_j \leq t_\mu < C_j \right\}$$

procedure GENERATE-PARAMETRIZED-ACTIVE-SCHEDULES (Π, Δ)
1 *Initialization*: $\mu := 1$; $t_1 := 0$; $\mathscr{A}_0 := \{0\}$; $\Gamma_0 := \{0\}$;
$\mathscr{C}_0 := \{0\}$; $R'_k(0) := R_k$, $(k \in \mathscr{R})$;
2 **while** $|\mathscr{C}_\mu| < n+2$ **repeat**
3 *Update:* $\mathscr{D}_\mu$;
4 **while** $\mathscr{D}_\mu \neq \{\}$ **repeat**
5 *Select activity with highest priority:*
$j^* := \underset{j \in \mathscr{D}_\mu}{\text{argmax}} \{ \Pi_j \}$;
6 *Calculate earliest finish time in terms of precedence only:*
$EC_{j^*} := \max_{i \in Pred(j)} \{C_i\} + p_{j^*}$;
7 *Calculate earliest finish time in terms of precedence and capacity:*
$C_{j^*} := p_{j^*} + \min \Big\{ t \in [EC_{j^*} - p_{j^*}, \infty] \cap \Gamma_\mu \mid r_{j^*k} \leq R'_k(\tau),$
$k \in \mathscr{R} \mid r_{j^*k} > 0,\ \tau \in [t, t + p_{j^*}] \Big\}$;
8 *Update:* $\mathscr{C}_\mu := \mathscr{C}_{\mu-1} \cup \{ j^* \}$; $\Gamma_\mu := \Gamma_{\mu-1} \cup \{ C_{j^*} \}$;
9 *Iteration increment:* $\mu := \mu + 1$;
10 *Update* $\mathscr{A}_\mu, \mathscr{D}_\mu, R'_k(t) \mid t \in [C_{j^*} - p_{j^*}, C_{j^*}],\ k \in \mathscr{R} \mid r_{j^*k} > 0$;
11 **end while**;
12 *Determine the time associated with the activity selected at iteration* μ*;*
$t_\mu := \min \{ t \in \Gamma_{\mu-1} \mid t > t_{\mu-1} \}$;
13 **end while**;
end GENERATE-PARAMETRIZED-ACTIVE-SCHEDULES;

Fig. 31.2 Pseudocode to generate parameterized active schedules

The remaining resource capacity of resource k at instant time t_μ is given by

$$R'_k(t_\mu) = R_k(t_\mu) - \sum_{j \in \mathscr{A}_\mu} r_{jk}$$

All activities that have been scheduled up to iteration μ are contained in the set $\mathscr{C}_\mu$ and Γ_μ denotes the set of finish times of the activities in $\mathscr{C}_\mu$. Let Δ_μ be the delay time associated with the activity being scheduled at iteration μ, and let the set $\mathscr{D}_\mu$ comprise all activities which are precedence-feasible in the interval $[t_\mu,\ t_\mu + \Delta_\mu]$, i.e.,

$$\mathscr{D}_\mu = \{ j \in V \setminus \mathscr{C}_{\mu-1} \mid C_i \leq t_\mu + \Delta_\mu \quad \forall i \in Pred(j) \}$$

The algorithmic description of the schedule-generation scheme used to generate parameterized active schedules is given by the pseudocode shown in Fig. 31.2.

The basic idea of parameterized active schedules is incorporated in the selection step of the procedure, i.e., in the step

$$j^* := \underset{j \in \mathscr{D}_\mu}{\text{argmax}} \{ \Pi_j \}$$

The set $\mathscr{D}_\mu$ forces the selection to be made only amongst activities which will have a delay smaller or equal to the maximum allowed delay.

Parameters Π_j (priority of activity j) and $\mathscr{D}_\mu$ (delay for the activity being scheduled at iteration μ) are supplied by the genetic algorithm.

31.6 Computational Results

In the next subsections we present the details of the computational experiments used to illustrate the effectiveness of the algorithm described in this chapter.

31.6.1 Test Problems

The test problems used in the computational experiments are the ones proposed by Gonçalves et al. (2008). These test problems have known optimal solutions equal to zero for the measure of performance described in Sect. 31.2 (i.e., *tardiness* $= 0$, *earliness* $= 0$, and *flow time deviation*$= 0$).

Five types of multi-project instances were used, with 10, 20, 30, 40, and 50 single-project instances, each with 120 activities. For each problem type, 20 instances were used. Since each single-project instance has 120 activities, we have that each multi-project instance has 1,200, 2,400, 3,600, 4,800, and 6,000 activities, respectively. Each activity can use up to four resources. The average number of overlapping projects in execution can be 3, 6, 9, 12, and 15. Table 31.3 shows the combinations of the number of overlapping projects used for the problems with 10, 20, 30, 40, and 50 single-projects.

31.6.2 BRKGA Configuration

Although there is no straightforward way to configure the parameters of a genetic algorithm, our past experience with genetic algorithms based on the same evolutionary strategy (see Gonçalves and Almeida 2002; Gonçalves and Resende 2004, 2011b, 2012, 2013, 2014; Gonçalves et al. 2005, 2008) has shown that good results can be obtained with the values of n_{elit}, n_{mut}, and Crossover Probability (π_{elit}) shown in Table 31.1.

Table 31.1 Range of parameters for the evolutionary strategy

Parameter	Interval
n_{elit}	$(0.10–0.25) \times \sigma_{pop}^{init}$
n_{mut}	$(0.15–0.30) \times \sigma_{pop}^{init}$
Crossover probability (π_{elit})	(0.70–0.85)

Table 31.2 Configuration of the BRKGA for the computational experiments

Population size:	min{0.2 × Number of activities in the multi-project, 250}
Crossover probability:	0.7
Selection:	The top 10 % from the previous population chromosomes are copied to the next generation
Mutation:	20 % of the population chromosomes are replaced with new randomly generated chromosomes
Fitness:	See Eq. (31.5)
Stopping criterion:	50 generations

For the population size we obtained good results by indexing it to the size of the problem, i.e., use small size populations for small problems and larger populations for larger problems. Having in mind this past experience and in order to obtain a reasonable configuration, we conducted a factorial analysis on a small pilot set of problem instances not included in the experimental tests. The configuration shown in Table 31.2 was the best in terms of the sum of fitness values and the number of best results and was held constant for all problem instances in the experiments. The experimental results demonstrate that this configuration not only provides high-quality solutions but it is very robust.

31.6.3 Results

Table 31.3 summarizes the experimental results. It lists the fitness, earliness, tardiness, and flow time deviation for each problem type. Let m be the number of projects in each problem instance. Averages and standard deviations were computed for the 20 problem instances included in each problem type. Columns Avg^1 and SD^1 list averages and standard deviations for the expression

$$\frac{1}{m}\left(w^T \sum_{i=1}^{m} T_i^3 \;+\; w^E \sum_{i=1}^{m} E_i^2 \;+\; w^{FD} \sum_{i=1}^{m} FD_i^2\right)$$

Columns Avg^2 and SD^2 list, respectively, averages and standard deviations for the expression

$$\frac{1}{m}\sum_{i=1}^{m} E_i$$

Columns Avg^3 and SD^3 list, respectively, averages and standard deviations for the expression

$$\frac{1}{m}\sum_{i=1}^{m} T_i$$

and columns Avg[4] and SD[4] list, respectively, averages and standard deviations for the expression

$$\frac{1}{m}\sum_{i=1}^{m} FD_i$$

The last column with heading *% Improv* represents the percentage improvement of the average last generation fitness values on those of the first generation, i.e.,

$$100\,\% \times \frac{(\text{Fitness at first generation} - \text{Fitness at last generation})}{\text{Fitness at first generation}}$$

Table 31.3 shows that all averages of the tardiness are close to zero and that the averages values of the earliness are also close to zero for all instances with more than three overlapping projects. As expected, the fitness obtained gets smaller (i.e., improves) as the number of overlappings of projects increases. This is due to the fact that as the number of overlapping projects increases, so does the flexibility in terms of capacity, therefore allowing for more possibilities of finding good schedules. Finally, the *% Improv* values show that the BRKGA achieves a substantial

Table 31.3 Experimental results

No	No	Fitness		Tardiness		Earliness		Flow dev.		No	%
Proj's	Overl.	Avg[1]	SD[1]	Avg[2]	SD[2]	Avg[3]	SD[3]	Avg[4]	SD[4]	Best	Improv
10	3	10.35	18.56	0.00	0.00	1.20	1.41	0.38	0.54	17	99.99
20	3	73.14	117.52	0.00	0.00	2.57	2.91	1.07	1.97	17	100.00
	6	0.95	2.10	0.00	0.00	0.42	0.27	0.03	0.07	20	100.00
30	3	210.13	202.81	0.01	0.02	3.92	2.88	1.74	1.45	18	100.00
	6	3.89	7.11	0.00	0.00	0.60	0.40	0.09	0.20	20	100.00
	9	0.48	0.37	0.00	0.00	0.38	0.12	0.02	0.05	19	100.00
40	3	1,324.14	1,282.69	0.06	0.06	9.45	7.29	6.15	4.77	15	100.00
	6	6.18	15.00	0.00	0.00	0.59	0.35	0.11	0.22	18	100.00
	9	4.48	16.52	0.00	0.00	0.50	0.25	0.06	0.21	18	100.00
	12	2.00	4.28	0.00	0.00	0.52	0.26	0.04	0.08	17	100.00
50	3	2,584.49	2,887.14	0.07	0.04	14.68	5.68	7.40	6.42	11	99.91
	6	25.87	57.23	0.00	0.00	0.87	0.60	0.23	0.39	17	100.00
	9	0.73	0.79	0.00	0.00	0.43	0.11	0.02	0.05	20	100.00
	12	1.35	2.16	0.00	0.00	0.50	0.17	0.02	0.05	18	100.00
	15	1.07	1.98	0.00	0.00	0.50	0.15	0.01	0.04	13	100.00

Table 31.4 Average elapsed time for 50 generations

Problem instance type (number of projects):	10	20	30	50
Average elapsed time (in seconds) for 50 generations:	178	449	840	1,860

improvement in the quality of the solutions. Sometimes the average percentage improvement is as large as 100 %.

The computational experiments were run on a PC with a 1.33 GHz AMD Thunderbird CPU on the MS Windows Me operating system and the algorithm was implemented in Visual Basic 6.0. Table 31.4 presents the average computational times, in seconds, for each problem instance and for 50 generations.

31.7 Conclusions

This chapter presents the Basic Multi-Project Scheduling Problem and a solution approach using a biased random-key genetic algorithm. The chromosome representation of the problem is based on random keys. The schedules are constructed using a schedule-generation scheme that generates parameterized active schedules based on priorities, delay times, and release dates generated by the biased random-key genetic algorithm.

The approach is tested on a set of test problems with 10, 20, 30, 40, and 50 projects (having 1,200, 2,400, 3,600, 4,800, and 6,000 activities, respectively). In the computational experiments, the algorithm obtained values near the optimum (zero), therefore validating the effectiveness of the proposed approach.

Acknowledgements This work has been partially supported by funds granted by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT, the Foundation for Science and Technology, project PTDC/EGE-GES/117692/2010.

References

Ash R (1999) Activity scheduling in the dynamic, multi-project setting: choosing heuristics through deterministic simulation. In: Proceedings of the 1999 winter simulation conference, Phoenix, pp 937–941

Baker KR (1974) Introduction to sequencing and scheduling. Wiley, New York

Bean JC (1994) Genetics and random keys for sequencing and optimization. ORSA J Comput 6:154–160

Błażewicz J, Lenstra JK, Rinnooy Kan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5:11–24

Bock D, Patterson J (1990) A comparison of due date setting, resource assignment, and job preemption heuristics for the multi-project scheduling problem. Decis Sci 21:387–402

Browning TR, Yassine AA (2010) Resource-constrained multi-project scheduling: priority rule performance revisited. Int J Prod Econ 126(2):212–228

Cai Z, Li X (2012) A hybrid genetic algorithm for resource-constrained multi-project scheduling problem with resource transfer time. In: Proceedings of the 2012 IEEE international conference on automation science and engineering (CASE 2012), Seoul, pp 569–574

Deckro R, Winkofsky E, Hebert J, Gagnon R (1991) A decomposition approach to multi-project scheduling. Eur J Oper Res 51:110–118

Drexl A (1991) Scheduling of project networks by job assignment. Manag Sci 37(12):1590–1602

Dumond J, Mabert V (1988) Evaluating project scheduling and due date assignment procedures: an experimental analysis. Manag Sci 34(1):101–118

Fendley L (1968) Towards the development of a complete multiproject scheduling system. J Ind Eng 19(10):505–515

Gonçalves JF, Almeida JR (2002) A hybrid genetic algorithm for assembly line balancing. J Heuristics 8:629–642

Gonçalves JF, Beirão NC (1999) Um algoritmo genético baseado em chaves aleatórias para sequenciamento de operações. Revista da Associação Portuguesa de Investigação Operacional 19:123–137 (in Portuguese)

Gonçalves JF, Resende MGC (2004) An evolutionary algorithm for manufacturing cell formation. Comput Ind Eng 47:247–273

Gonçalves JF, Resende MGC (2011a) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17:487–525

Gonçalves JF, Resende MGC (2011b) A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. J Comb Optim 22:180–201

Gonçalves JF, Resende MGC (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput Oper Res 39:179–190

Gonçalves JF, Resende MG (2013) A biased random key genetic algorithm for 2D and 3D bin packing problems. Int J Prod Econ 145(2):500–510

Gonçalves JF, Resende MG (2014) An extended akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. Int Trans Oper Res 21(2):215–246

Gonçalves JF, Sousa PSA (2011) A genetic algorithm for lot sizing and scheduling under capacity constraints and allowing backorders. Int J Prod Res 49:2683–2703

Gonçalves JF, Mendes JJM, Resende MGC (2005) A hybrid genetic algorithm for the job shop scheduling problem. Eur J Oper Res 167:77–95

Gonçalves JF, Mendes JJM, Resende MGC (2008) A genetic algorithm for the resource constrained multi-project scheduling problem. Eur J Oper Res 189:1171–1190

Krüger D, Scholl A (2010) Managing and modelling general resource transfers in (multi-) project scheduling. OR Spectr 32(2):369–394

Kumanam S, Raja K (2011) Multi-project scheduling using a heuristic and memetic alogrithm. J Manuf Sci Prod 10(3–4):249–256

Kurtulus I (1985) Multiproject scheduling: analysis of scheduling strategies under unequal delay penalties. J Oper Manag 5(3):291–307

Kurtulus I, Davis E (1982) Multi-project scheduling: categorization of heuristic rules performance. Manag Sci 28:161–172

Kurtulus I, Narula S (1985) Multi-project scheduling: analysis of project performance. IIE Trans 17:58–66

Lawrence S, Morton T (1993) Resource-constrained multi-project scheduling with tardy costs: comparing myopic bottleneck and resource pricing heuristics. Eur J Oper Res 64:168–187

Lova A, Tormos P (2001) Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. Ann Oper Res 102(1):263–286

Lova A, Tormos P (2002) Combining random sampling and backward-forward heuristics for resource-constrained multi-project scheduling. In: Proceedings of the eight international workshop on project management and scheduling, Valencia, pp 244–248

Lova A, Maroto C, Tormos P (2000) A multicriteria heuristic method to improve resource allocation in multiproject scheduling. Eur J Oper Res 127:408–424

Mendes J (2003) Sistema de apoio à decisão para planeamento de sistemas de produção do tipo projecto. Ph.D. dissertation, Universidade do Porto, Porto (in Portuguese)

Mohanthy R, Siddiq M (1989) Multiple projects multiple resources-constrained scheduling: some studies. Int J Prod Res 27(2):261–280

Özdamar L, Ulusoy G, Bayyigit M (1998) A heuristic treatment of tardiness and net present value criteria in resource constrained project scheduling. Int J Phys Distrib Logist Manag 28:805–824

Pritsker A, Allan B, Watters L, Wolfe P (1969) Multiproject scheduling with limited resources: a zero-one programming approach. Manag Sci 16:93–108

Shankar V, Nagi R (1996) A flexible optimization approach to multi-resource, multi-project planning and scheduling. In: Proceedings of 5th industrial engineering research conference, Minneapolis, pp 263–267

Spears WM, Dejong KA (1991) On the virtues of parameterized uniform crossover. In: Proceedings of the fourth international conference on genetic algorithms, San Diego, pp 230–236

Talbot FB (1982) Resource-constrained project scheduling with time-resource tradeoffs: the nonpreemptive case. Manag Sci 28(10):1197–1210

Tsubakitani S, Deckro R (1990) A heuristic for multi-project scheduling with limited resources in the housing industry. Eur J Oper Res 49:80–91

Vercellis C (1994) Constrained multi-project planning problems: a Lagrangean decomposition approach. Eur J Oper Res 78:267–275

Woodworth BM, Willie CJ (1975) A heuristic algorithm for resource leveling in multi-project, multi-resource scheduling. Decis Sci 6(3):525–540