Computers & Industrial Engineering 107 (2017) 39–56

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Scheduling multiple, resource-constrained, iterative, product development projects with genetic algorithms

Ali A. Yassine [a,*], Omar Mostafa [a], Tyson R. Browning [b]

[a] Department of Industrial Engineering & Management, American University of Beirut, Beirut, Lebanon
[b] Neeley School of Business, Texas Christian University, TCU Box 298530, Fort Worth, TX 76129, United States

ARTICLE INFO

Article history:
Received 15 August 2016
Received in revised form 22 January 2017
Accepted 1 March 2017
Available online 3 March 2017

Keywords:
Resource-Constrained Project Scheduling Problem (RCPSP)
Resource-Constrained Multi-Project Scheduling Problem (RCMPSP)
RCMPSP with Feedback (RCMPSPwF)
Iteration
Rework
Design Structure Matrix (DSM)
Genetic Algorithm (GA)

ABSTRACT

Many product development (PD) projects rely on a common pool of scarce resources. In addition to resource constraints, there are precedence constraints among activities within each project. Beyond the feed-forward dependencies among activities, in PD projects it is common for feedback dependencies to exist that can result in activity rework or iteration.

In such a multi-project, resource-constrained, iterative environment, this paper proposes two new genetic algorithm (GA) approaches for scheduling project activities. The objective is to minimize the overall duration of the portfolio of PD projects. These proposed GAs are tested on sample scheduling problems with and without stochastic feedback. We show that these algorithms provide quick convergence to a globally optimal solution.

Furthermore, we conducted a comparative analysis of the proposed GAs with 31 published priority rules (PRs), using test problems generated to the specifications of project, activity, and resource-related characteristics such as network density (complexity), resource distribution, resource contention, and rework probability (amount of iteration). The GAs performed better than the PRs as each of these factors increased. We close the paper by providing managers with a decision matrix showing when it is best to use the published PRs and when it is best to use the GAs.

1. Introduction

The timely delivery of new products and services is critical for the success and survival of organizations. Due to an increase in market competition, fast technological advancements, as well as evolving customers' needs and impatience, it has become very important to improve the efficiency with which projects are completed and new products are brought to market (Hendricks & Singhal, 1997; Herm, 2013; Hum & Sim, 1996; Majava, Haapasalo, Belt, & Mottonen, 2013). Moreover, many organizations are faced with the challenge of managing the simultaneous execution and management of a portfolio of development projects under tight time and resource constraints (Beaujon, Marin, & McDonald, 2001; Pennypacker & Dye, 2002; Rad & Levin, 2006). In such an environment, project management and scheduling skills become very critical to the organization. Herroelen (2005) mentioned that multi-project environments are quite common in project scheduling practice and offer many future research opportunities. He added that a large number of projects are carried out in a multi-project environment, and thus even a small improvement in their management will yield a large benefit to the project management field.

Project cost and schedule overruns have persisted for decades, in spite of numerous advances in the field of project management (Anderson & Tucker, 1994; Kerzner, 2013; PMI, 2013). Starting with the Project Evaluation and Review Technique (PERT) and the critical path method (CPM), network techniques have continued to evolve and advance (Mantel, Meredith, Shafer, & Sutton, 2007; Spinner, 1989). Advances include resource-constrained scheduling, resource leveling, and probabilistic risk assessments (Demeulemeester, 2002; Schwindt & Zimmermann, 2014). Additionally, alternative approaches to product and software development management such as the waterfall and spiral methods have been proposed (Unger & Eppinger, 2009).

The resource-constrained project scheduling problem (RCPSP) presents an extension to the standard CPM and PERT techniques by including the availability of resources during scheduling. However, an organization may often have several concurrent projects. While the projects may otherwise be unrelated, they depend on a common pool of resources. An overwhelming theme in the

* Corresponding author.
E-mail addresses: ali.yassine@aub.edu.lb (A.A. Yassine), omarmm19@gmail.com (O. Mostafa), t.browning@tcu.edu (T.R. Browning).

http://dx.doi.org/10.1016/j.cie.2017.03.001
0360-8352/

multi-project management literature is the issue of resource allocation between simultaneous projects (Engwall & Jerbrant, 2003). This extension is known as the resource-constrained multi-project scheduling problem (RCMPSP) (Kolisch and Padman, 2001).

RCPSPs and RCMPSPs are both strongly NP-hard, so there are no known algorithms for finding optimal solutions in polynomial time (Lenstra & Kan, 1978). Hence, most researchers have sought efficient heuristic and meta-heuristic techniques. Priority rule (PR) heuristics are the most common heuristics considered for very large problems, and are known for their speed, simplicity and ability to construct initial solutions (Browning & Yassine, 2010b). Several meta-heuristics have also been used, including simulated annealing, Tabu search, genetic algorithms, and ant colony optimization (Hartmann & Kolisch, 2000; Kolisch & Hartmann, 1998, 2006).

To complicate things further, in product development (PD) projects, scheduling activity iteration or rework has always been a challenge. Kang and Hong (2009) noted that the delays caused by activity iteration in a multi-project environment are as significant as those resulting from resource constraints. Even though the occurrence of iterations may not be known with certainty prior to project execution, a skilled manger can identify many potentially iterative activities and plan accordingly. Understanding the web of information flow in a project can help identify potential iterations (Eppinger, 2001). Unfortunately, many managers fail to plan for iterations in advance and integrate them into project schedule and cost estimation (Browning & Eppinger, 2002).

The main objectives of this paper are (1) to find an optimal or near-optimal duration distribution for a RCMPSP in the presence of stochastic activity iterations and (2) to compare this result to the performance of PRs. To achieve the first objective, this paper introduces two new genetic algorithm (GA) -based approaches. The procedure is based on a modified genetic encoding of a standard simple GA and the tailoring of its operators to suit this problem. To achieve the second objective, we utilized a full factorial experiment with randomly generated problem instances to demonstrate the superiority of the proposed GA-based approach by comparing our results to published PRs. We found significant differences in the performance of the GAs relative to the PRs in cases of high levels of iteration, network density (complexity), and resource constraints. Moreover, the GAs showed better convergence towards optimal solutions (shortest duration), especially in high-complexity and iterative projects. Finally, we organize these results for managers, distinguishing between the project and portfolio management perspectives.

The rest of the paper is organized as follows. In Section 2 we describe the relevant literature in traditional project management, design structure matrix (DSM), and GAs. Section 3 describes the two proposed GA approaches (Sampling and Variable-length GAs), discussing all parameters and methodologies and giving managers a guideline for scheduling decisions. Section 4 discusses the implementation of the problem, and Section 5 introduces the data used in the analyses and calibration of the model via sensitivity analyses of GA parameters and network characteristics. It also discusses model validation according to published benchmark studies of Kolisch and Sprecher (1997) and Browning and Yassine (2016). Furthermore, Section 5 describes the setup of and computational results from the comparison of the GAs with PRs. Section 6 summarizes the paper and draws final conclusions.

2. Literature review

The literature review covers both the DSM method and the RCMPSP. However, within this vast literature, we focus on meta-heuristic approaches, genetic algorithms (GAs) in particular. From this review, it will be noticed that DSM-based simulation techniques have not yet considered schedule optimization, nor have GA-based scheduling problems considered iteration (i.e. cyclic project networks such as PD), when dealing with RCPSPs. So, our paper fills this particular gap in the project scheduling literature.

2.1. Design structure matrix (DSM)

A DSM is an efficient and commonly used method of showing the relationships among the activities in a project (Yassine & Braha, 2003). Given a set of n activities in a project, the corresponding DSM is an $n \times n$ matrix where the project activities are the row and column headings listed in the same order. The precedence relationships among activities appear in the off-diagonal elements of the matrix. If activity j depends on activity i (that is, i feeds j), then the value of element ij (column i, row j) is one (or flagged with a mark such as "X") in a binary DSM. Otherwise, the value of the element is zero (or left empty) (Yassine, 2004). If the activities are executed in the same order as they appear in the DSM, then marks below the diagonal represent forward information from activity i to j, while those above the diagonal represent feedback information from activity j to i.

Feed-forward dependencies are typical to deal with, but feedback dependencies are more challenging from a scheduling point of view. The latter dependencies exist due to the uncertainty in performing some activities with a lack of information. As new information emerges throughout the development process, this uncertainty is resolved and may either validate how activities were performed or reveal some mistakes that require (first-order) rework, repetition, or iteration. Rework in an upstream activity, caused by downstream activity, can also cause *second-order rework* (Browning & Eppinger, 2002) due to the cascading of changes through interim activities. Thus, the super-diagonal marks represent the probability of iteration (returning to previous activities), while the sub-diagonal marks note the probability of second-order rework (following any first-order rework). The DSM that contains such rework probabilities is called a Probability DSM. Browning and Eppinger (2002) also introduced an Impact DSM, which includes impact values (between 0 and 1) representing the rework percentages of an activity's initial duration.

Several DSM-based simulation models (and tools) exist in the literature (e.g., Abdelsalam & Bao, 2006; Browning & Eppinger, 2002; Zhuang & Yassine, 2004). Although all of these models are mainly aimed at determining the process completion time and cost for a given task network structure (process architecture), some also have an added component for determining the optimal architecture (e.g., Abdelsalam & Bao, 2006; Kang & Hong, 2009; Zhuang & Yassine, 2004) by testing (i.e., calculating the time and cost of) various architectural arrangements. Abdelsalam and Bao (2006) presented a simulation-based optimization framework that determines the sequence of activity execution in a PD project with minimal total duration based on an "iteration factor" representing coupling strength and the number of iterations required for convergence. Zhuang and Yassine (2004) sampled several potential iterative scenarios from an existing DSM model, where each scenario represents a project without feedback. Then, these projects were scheduled using a simple GA scheduling algorithm. However, none of these DSM-based simulation/optimization models have accounted for resource constraints across multiple projects.

2.2. Genetic algorithms (GAs)

Metaheuristic algorithms became an important part of modern optimization (Goldberg, 1989; Holland, 1975). A wide range of metaheuristic algorithms have emerged over the last two decades, and many metaheuristics such as genetic algorithms (GA) and

particle swarm optimization (PSO) are becoming increasingly popular (Gendreau & Potvin, 2010). These algorithms have been used in various engineering applications from job shop and project scheduling (Gonçalves, Mendes, & Resende, 2005 and Gonçalves, Mendes, & Resende, 2008), to Crashworthiness and NVH Optimization, structural design optimization (Gökdağ & Yildiz, 2012; Kiani & Yildiz, 2016). However the use of GAs has been more prevailing in the in project scheduling literature than any other metaheuristic technique; especially, for DSM analysis (Kolisch & Hartmann, 1998; Meier, Yassine, & Browning, 2007).

In brief, a simple GA (SGA) for project scheduling works as follows. A feasible schedule of the project's activities is encoded into a *chromosome*. Multiple chromosomes form a GA *population*. By selecting the fittest chromosomes (i.e., the ones with the best result according to an objective or fitness function, such as project duration) and applying the ordinary genetic operators—selection, crossover, and mutation—the population is expected to improve over time (i.e., consist of shorter and shorter duration projects). The GA proceeds until a predefined convergence criterion is reached. Fig. 1 depicts this flow for the SGA.

The use of GAs in process DSM analysis has been a topic of interest in numerous studies (Browning, 2016). Early works include Rogers (1996), McCulley and Bloebaum (1996), and Altus, Kroo, and Gage (1996). All these essentially employed a SGA (with little tailoring of the various GA operators) that supports DSM resequencing in accordance with user-defined objectives. Later, Whitfield, Duffy, Coates, and Hills (2003) examined GA crossover and mutation operators for sequencing DSMs, finding that earlier publications acknowledged the efficacy of GAs in combinatorial problems but did not exploit their entire repertoire. Accurate information about the setting of GA parameters and information about the incentive behind the use of certain GA operators was absent or only based on incomplete tests when present. Critical improvements in GAs such as hybridization or niching techniques were missing. In addition, since "competent GAs" had not yet been utilized for DSM sequencing, an advanced investigation was critical.

The optimality of a GA solution depends on the goal of DSM resequencing. In an activity ordering context, goals comprise one or more of the following: minimizing iteration/feedback, maximizing concurrency, and diminishing development lead-time (project duration) and cost. Meier et al. (2007) applied GAs to a binary DSM model to define an optimal sequence for a set of design activities, finding that DSM characteristics (e.g., size, sparseness, and sequencing objective) can cause grave problems for SGA designs. To manage this deficiency, they explored the use of a "competent" GA, the ordering messy GA (OMeGA), and confirmed its superiority over an SGA for hard DSM problems. Lancaster and Cheng (2008) suggested the need to increase the mutation probability from one generation to the next to increase the chances for a GA to ultimately converge on a global optimum. Finally, Kang and Hong (2009) used a GA in conjunction with a DSM model to simulate iteration and resource constraints in a multi-project environment, but their model is also concerned with sequencing activities rather than project scheduling. That is, they assumed sequential execution of activities, while parallel execution is prohibited.

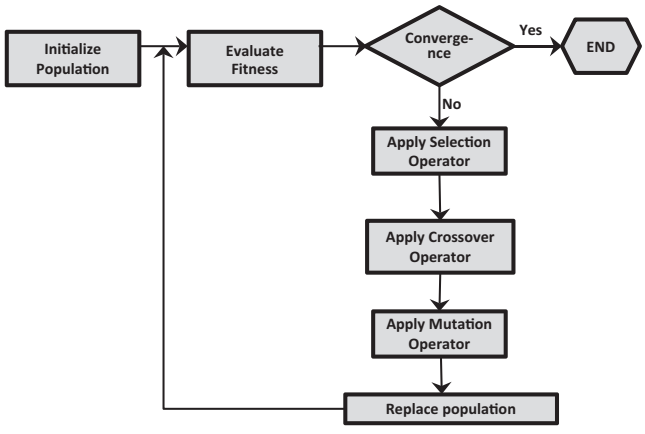


Fig. 1. Flowchart of SGA.

Resource constraints can also be prioritized using human evaluation if an interactive genetic algorithm is utilized (Kosorukoff, 2001). However, the challenge remains for IGA to reduce user fatigue during the evaluation process and also to reduce the noise in the process of evolution. To address the issue, a multi-stage interactive genetic algorithm (MS-IGA) has been proposed, which divides the large population of the traditional interactive genetic algorithm (TIGA) into several stages according to different functional requirements (Dou, Zong, & Li, 2016b; Dou, Zong, & Nan, 2016a).

2.3. Resource-Constrained (Multi) Project Scheduling Problems: RCPSP and RCMPSP

Kolisch and Padman (2001) surveyed a number of techniques developed for the RCPSP, including dynamic programming, zero-one programming, and implicit enumeration with branch and bound. Some examples of exact solution methods can be found in Demeulemeester and Herroelen (1992, 1996), and Sprecher (1996). Among them, the branch and bound approach is the most widely applied. However, its depth-first or breadth-first searches cannot exhaustively explore a large-scale project scheduling problem. Many heuristic-based solution approaches can also be found in the literature (Browning & Yassine, 2010b).[1] There is not a single priority rule that performs well all the time; however, Browning and Yassine (2010b) suggested that different rules must be used based on different project and problem characteristics. Finally, many different meta-heuristic approaches have been developed to solve intractable problems quickly, efficiently, and fairly satisfactorily. Surveys of such approaches can be found in the literature (Brucker, Drexl, Möhring, Neumann, & Pesch, 1999; Ip, Li, Man, & Tang, 2000; Kolisch & Hartmann, 1998, 2006). With a special focus on a single-project (i.e., the RCPSP), Hartmann (1998, 2002) developed a competitive GA with permutation-based encoding and introduced a self-adapting representation scheme that automatically determines the best schedule decoding procedure. Gonçalves, Mendes, and Resende (2008) used a GA approach for the RCMPSP based on a random key chromosome encoding and a schedule generation procedure which creates so-called parameterized active schedules. Valls, Ballestin, and Quintanilla (2008) proposed a hybrid GA tailored to the RCPSP with a specific crossover and local search operator. However, these meta-heuristic approaches have not yet been applied to PD projects with iterative networks.

3. Problem description

In this section, we introduce the RCMPSP with feedback (RCMPSPwF). We consider a portfolio of resource-constrained projects, where each project is represented by its probability and impact DSMs. Each activity in a project is defined by the following features:

- Stochastic duration: each activity has a triangularly distributed duration (minimum, most likely, maximum).

[1] Most commonly used rules are First Come First Serve (FCFS), Shortest Operation First (SOF), and Minimum Slack (MINSLK) (Browning & Yassine, 2010b).

- Maximum number of reworks: each activity has a pre-specified maximum possible number of reworks.
- Resources: each activity requires specific resources defined by name and quantity.
- Learning factor: as some activities are reworked, their duration decreases due to a learning curve.

A portfolio consists of L projects and a project consists of N_l activities labeled $i = \{1,\ldots,N_l\}$. The total number of activities in the portfolio becomes $m = \sum_{l=1}^{L} N_l$. For every activity i (in project l), the duration is denoted by d_{il} and it takes three values—WCV_{il}, MLV_{il}, and BCV_{il}—the worst case (pessimistic) value, most likely value, and best case (optimistic) value, respectively. Precedence relations among activities are given by sets of immediate predecessors P_{il}, indicating that an activity i (in project l) may not begin until all of its predecessors are completed. That is, $t_{jl} + d_{jl} \leq t_{il}$, if j is the predecessor of activity i (in project l) and for all $j \in P_{il}$, where t_{il} is the starting time of activity i (in project l).

Although projects may be unrelated by precedence constraints, they depend on a common pool of resources and are therefore related by resource constraints. We consider a set of renewable resources where the per-period usage of resource k by activity i in project l is written as r_{ilk}. Any two resource-conflicting activities cannot have overlapping time while being performed; that is, $t_{jl} + d_{jl} \leq t_{il}$ or $t_{jl} \leq t_{il} + d_{il}$, if $r_{ilk} = r_{jlk}$. We assume that a resource must be devoted to an activity until it is completed before beginning another activity (i.e., preemption is not allowed). Moreover, we assume the single-mode case, where a single resource type is assigned for performing an activity and the required resource is fixed.

Feedback is not a definite event; it depends on the rework probabilities. Initiation of a feedback cycle occurs according to the DSM probabilities, and the magnitude of rework for a specific activity is defined in the impact DSM and the learning factor. The learning factor for an activity is IC_{il}, a predefined improvement curve associated with each activity representing its rework duration as a percentage of its original duration: $(1 - IC_{il}) \times$ (original duration of activity).

Finally, although many objectives can be used for this problem, including minimizing the average delay for the project portfolio, maximizing the net present value (NPV), and maximizing resource leveling, we choose to minimize the total duration of the project portfolio such that both the precedence and resource constraints are respected (Kolisch & Padman, 2001). That is:

$$\text{Minimize } T = \max\{t_{il} + d_{il}\}, \quad \forall i \in N_l \text{ and for } l = 1,\ldots,L.$$

When dealing with the RCMPSP, two approaches have been used: (1) a single-project approach, using dummy precedence arcs to combine the projects into a single megaproject, thereby reducing the RCMPSP to a RCPSP with a single critical path, or (2) a multi-project (MP) approach, maintaining the RCMPSP and a separate critical path per project (Kurtulus & Davis, 1982). Here we use the first approach because it allows us to model and solve both the RCPSP and RCMPSP using a single DSM and apply the same proposed methods to both types of project scheduling problems (Yassine, 2002). As a result, we treat the project portfolio as a large single project associated with the combined DSM as shown in Fig. 2. In this figure, the three separate projects, represented by three separate DSMs on the left of the figure, were combined to form a single DSM for a single mega-project (to the right) (Yassine & Browning, 2002).

4. Solution methodology

In this section, we describe the general mechanics of two proposed solution approaches for the RCMPSP with feedback (RCMPSPwF): sampling GA (SaGA) and variable-length GA (VLGA). The SaGA is based on the general approach proposed by Zhuang and Yassine (2004); however, in this paper it is detailed and formalized. The second approach, VLGA, is new and based on a novel method to operate on chromosomes of variable lengths. Both are based on a SGA approach, Hartmann's (1998) permutation-based GA.

An activity sequence is assumed to be a precedence-feasible permutation of the set of activities when each chromosome (genotype) is related to a uniquely determined schedule (phenotype). This schedule of activities is defined in the order that is prescribed by the sequence so that each activity is assigned to a set of predecessors and a feasible start time. This introduces a redundancy in the search space as distinct elements of the search space (i.e., genotypes) may be related to the same schedule. Exchanging some activities in an activity sequence, we obtain a different feasible genotype. However, both genotypes are related to the same schedule.[2] Complex relationships between activities in a RCMPSPwF make it nearly impossible to have a feasible schedule or a fit chromosome by random generation. Dealing with rework in a RCMPSP introduces probabilistic activities to a sequence, which makes precedence-feasible permutation problematic too. Having new precedence constraints according to rework makes it preferable to fit the individuals and sequences based on precedence only, so that all possible schedules will be considered. The fitness value of chromosomes (feasible sequences) is calculated considering resource constraints and the possibility of parallel activities. This gives a broader range of possible sequences than merely assigning each activity to its earliest feasible start time.

4.1. Basic GA constructs

Chromosome Representation: Each chromosome represents a feasible sequence of activities, which must be completed to determine the overall duration. It contains all associated activities whether they belong to a single project or to multiple projects. Fig. 3 shows how different length chromosomes can result due to rework. For example, if no rework was initiated after the completion of activity 4, then chromosome number 1 is produced. Chromosome number 2 is the result of reworking activity 2 (first order rework, but no other second order reworks) after the completion of activity 4. The letter "f" attached to some activity numbers in the various chromosomes means that the activity is repeated once due to feedback; "ff" means that the activity is repeated twice and so on. Hence, chromosome length may vary due to the possibility of rework. We call the activities without any letter attached to them "original" activities and all other activities "reworked" activities. Reworked activities are denoted by a number followed by at least one "f".

Initialization: Because highly-constrained scheduling problems have a small feasible search space, purely random generation of activity sequences results in a large number of infeasible solutions. Therefore, we use a permutation-based simulation procedure to produce an initial population of precedence-feasible chromosomes (Hartmann, 1998):

1. Randomly select an activity from the unselected activity pool and check if its immediate predecessor(s) are already selected. If not, continue this random selection until such an activity is found.

[2] That is, in some cases different activity sequences may refer to the same schedule; particularly if we have parallel activities. This means A1-A2-A3-A4-A5-A6 may end up to be the same schedule as A1-A3-A2-A4-A5-A6 if A2 and A3 are executed in parallel.

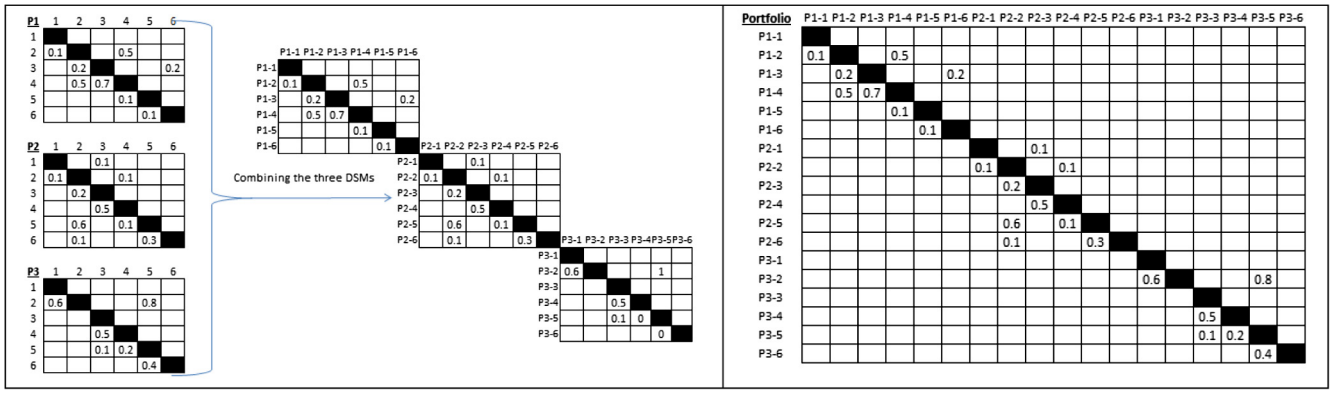


Fig. 2. Multi-project scheduling problem using a single DSM.

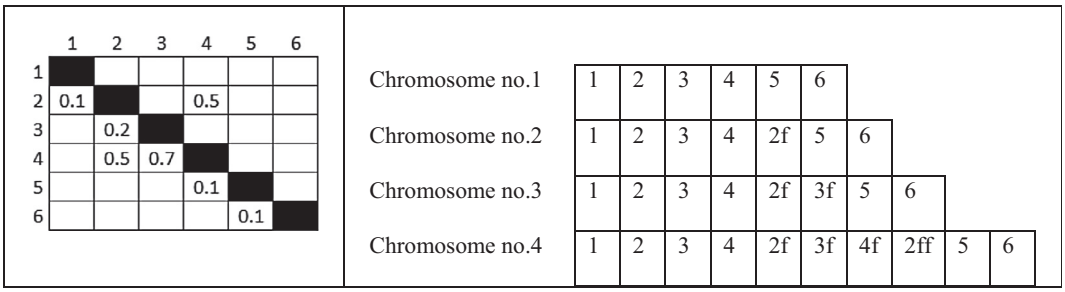


Fig. 3. Example chromosomes of different lengths due to potential rework.

2. Repeat step 1 until the set of unselected activities is empty, thereby generating a complete chromosome.
3. Repeat steps 1 and 2 until a number of chromosomes equal to the population size (N) has been generated.

Fitness evaluation of members of the population is based on project (or portfolio) duration. The earliest starting time is set for each activity to obtain the duration of all parallel projects in each chromosome of the population. Based on the sequence of activities in the chromosome representation, limited resources are allocated to activities as they become available:

1. Initialize the available time of all resources to zero.
2. Starting with the first activity index listed in the chromosome, compute for every activity: $T_i = \text{Max}\ \{t_l + d_l, \max(s_j + d_j)\}$, where l is the index of the activity last performed by the needed resource, and j is the index of the predecessor activity.
3. Repeat steps 1 and 2 until all activities have been computed, thereby providing the duration of the chromosome.
4. The fitness of each chromosome is the reciprocal of its duration, such that longer duration implies lower fitness.

Selection is the process of choosing one or two parents from the population for reproduction through crossover and mutation (Goldberg, 1989). The result is a mating pool. In general, the higher the fitness value of a chromosome, the greater its chance of being selected for reproduction. Both roulette and tournament selections are used in our model. The selection of chromosomes for crossover via roulette selection occurs according to proportional fitness. The probability of individual i to be selected is given by $p_i = \frac{F_i}{\sum_{i=1}^{N} F_i}$, where F_i is the fitness of a chromosome and N is the population size. In some cases, especially when activity durations are short, chromosomes could have equal or nearly-equal fitness, which makes it hard to select the fittest. In such cases, we move to a triple-based tournament selection strategy, where three chromosomes are randomly selected from the population and the one with the highest fitness is selected.

Crossover is the process of (randomly) taking two fit individuals from the mating pool and intermingling their chromosomes to create two "children" for the next generation (Goldberg, 1989). After choosing a mating parent chromosome, a random number is generated between 0 and 1. If this random number is less than the crossover factor, then the crossover is performed; otherwise, the chromosome survives to the next generation as it is. When the crossover proceeds between two mating chromosomes, another random number is generated to determine a crossover point in the chromosome genes. Either one-point or two-point crossover can be used in our model. With the one-point crossover operator, once the crossover point has been determined, the parts of the chromosomes to the right of the point are exchanged, producing new chromosomes. Alternatively, with the two-point crossover operator, two crossover points are determined, and the parts of the chromosomes in between those points are exchanged, producing new chromosomes. Each chromosome selected for crossover is already a competent chromosome for survival, which is why we should maintain a high probability that this chromosome will survive as it is. This means the crossover factor should be high, but not so high as to prevent a chance for the immediate survival of fit chromosomes (and to increase overall computational time). This is why it is commonly set between 0.6 and 0.8 (Hartmann, 1998; Lancaster & Ozbayrak, 2007).

Mutation is the process whereby a chromosome is deliberately changed to maintain diversity in the population. Each of the resulting chromosomes (from crossover) is subjected to a mutation process. A set of random numbers between 0 and 1 is generated through the chromosome's genes, and mutation alters one or more genes where the random number is less than the mutation factor. However, because mutation should not overly alter fit or optimal chromosomes, the mutation factor should be relatively

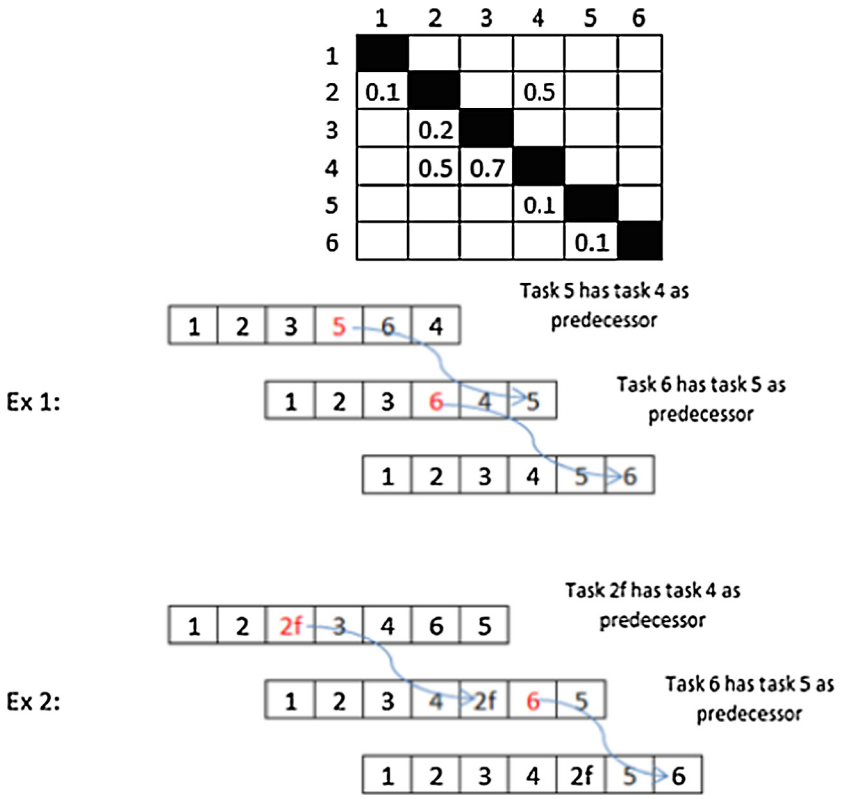


Fig. 4. Permutation process.

small (e.g., [0.01,0.1]) (Hartmann, 1998; Lancaster & Ozbayrak, 2007).

Instantaneous Feasibility Check (IFC): In both proposed GA approaches, we introduce a new operator called an instantaneous feasibility check (IFC) to ensure the feasibility of chromosomes after crossover and mutation. Any set of activities represented in a chromosome may be feasible or infeasible according to the following constraints: precedence constraints, all original activities exist, and no duplicates exist. The IFC detects any infeasibilities and repairs the chromosome to render it feasible and applicable. The IFC is applied to all chromosomes in each generation to ensure a feasible population:

1. Check if any duplicate activity exists in the chromosome.
2. Check if all the original activities exist in the chromosome.
3. Replace (a) a duplicate activity or (b) the highest-level reworked activity (i.e., an activity with most f's attached to it) in the chromosome with any missing original activities.
4. Ensure fulfillment of precedence constraints through a permutation procedure (by shifting an activity by the minimal necessary number of positions to the right), as follows:
   a. Move through the chromosome from left to right, checking that each activity follows its predecessors.
   b. If not, then shift the activity to a position after all of its necessary predecessors.

Fig. 4 demonstrates step 4. In example 1 (in Fig. 4), from left to right, activity 2 is a successor for activity 1 and activity 3 is a successor for activity 2. Activity 5 then appears before its predecessor (activity 4) so the latter activities are swapped. Then, the same thing happens for activities 6 and 5. In example 2 (Fig. 4), activity 4 feeds back to activity 2, which means activity 2f is a reworked activity which shows before its predecessor activity 4. This is why activity 2f is shifted after its predecessor.

As a demonstration of the IFC operator, consider the DSM of Fig. 4 and the corresponding examples in Fig. 5. Example 1 (in Fig. 5) shows a chromosome in which activity 4 is duplicated. IFC detects that task 5 is missing from the original activities and inserts it in place of the duplicate. Example 2 (in Fig. 5) shows a chromosome missing task 5 but has no duplicated task in the sequence. IFC leads to the insertion of task 5 instead of the latest reworked task which is 3ff which will not affect the chromosome feasibility since 3ff is not a predecessor for any other task in the chromosome.

This IFC has advantages and drawbacks. By ensuring feasible chromosomes, it reduces the number of generations required to reach an optimal or near-optimal solution. On the downside, it may lock the GA into a suboptimal solution by causing some feasible solutions to become extinct. Another disadvantage is that the generation will no longer be purely random. Moreover, applying the IFC operator to a large population increases the computational time per generation. However, a feasible population is essential for effective and efficient GA application.

4.2. Sampling GA (SaGA)

Starting with a RCMPSPwF, this approach randomly generates several representative RCMPSPs *without* feedbacks, each representing a potential instance of the original problem. We refer to this as a problem instance; the set of these instances is called a sample. The cardinality of this sample set is called the sample size, *S*, and defines the number of instances to be generated that is predefined and different from the population size, *N*. *S* should be large enough to properly represent the original RCMPSPwF but not to unduly increase the GA execution time (see Fig. 10 in Section 4.3 for further discussion). For example, consider the DSM in Fig. 4. One problem instance includes six activities without rework. Another problem instance includes one reworked activity, 2f. Applying

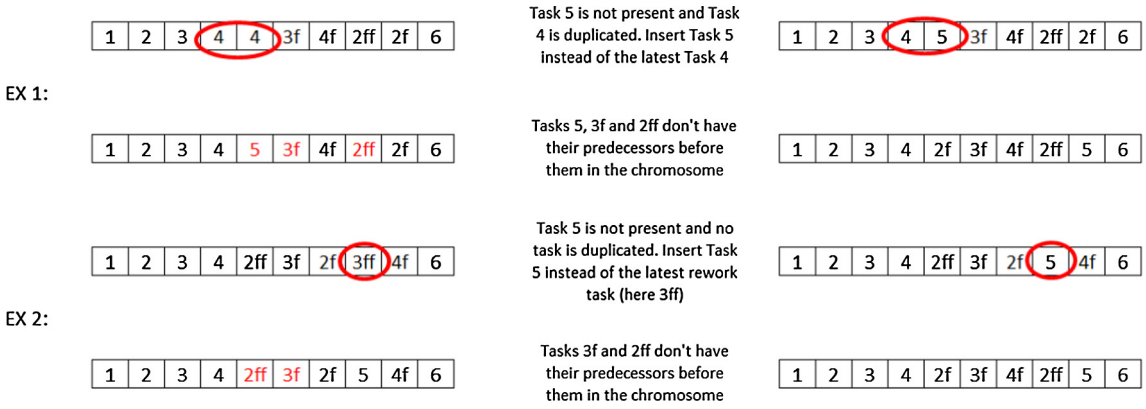


Fig. 5. Examples of applying the Instantaneous Feasibility Check (IFC) operation.

the GA to each problem instance separately results in a set of optimal solutions. The SaGA procedure proceeds as follows:

1. Generate the sample set of predefined size, *S*. Generate a large initial population of different chromosome lengths, where each chromosome represents a portfolio of projects under a particular feedback scenario.
2. Each chromosome of the initial population is then considered as a separate multi-project portfolio consisting of a set of original activities plus others appearing due to feedback-triggered rework. The latter activities are henceforward treated as regular activities with defined properties (duration, predecessors, and resources).
3. Each portfolio instance is treated as a separate problem and undergoes a SGA process (Fig. 1) as if no feedback exists.
4. The best schedule found for each problem is used to form a distribution of outcomes for the portfolio.

In summary, the inputs to this model are: the rework probability and impact DSMs, the activity definitions (name, resource requirements, duration, maximum number of reworks, and learning factor), the sample size (*S*), the number of generations (*G*), the population size (*N*), the crossover factor (*C*), and the mutation factor (*X*). Once these input parameters are defined (later in the paper we discuss how to set these parameters), each problem instance undergoes the following SGA process:

1. Define an initial population (initialization), a set of possible combinations of activities. The number of combinations or chromosomes is called the population size (*N*).
2. Adjust chromosomes through permutation (IFC step 4). Because precedence constraints render most randomly-generated chromosomes infeasible, chromosomes must be adjusted.
3. Calculate the fitness of each chromosome as the reciprocal of its duration.
4. Select chromosomes for crossover via roulette or tournament selection.
5. Perform crossover between selected chromosomes.
6. Perform mutation on the resulting chromosomes.
7. Repeat steps 2–4 for the chromosomes resulting from crossover and mutation chromosomes before joining the next generation.
8. The crossover and mutation process proceeds until a new generation with the defined population size (*N*) is produced.
9. Repeat steps 5–9 until the *G*th generation, which will contain chromosomes with the optimal or near-optimal duration and project schedule for this problem.

After applying the GA to all the projects in the sample, the fittest chromosome from the last generation of each problem is selected as the best solution. That is, each scenario from the sample will result in an optimal or near-optimal solution. As a result, we will obtain a frequency distribution for the *S* solutions, from which we infer the relative likelihood (probability) of specific duration outcomes (whole iterative problem/portfolio).[3]

4.3. Variable-Length GA (VLGA)

The VLGA approach deals with stochastic rework (i.e., DSM feedback marks) by enlarging the original set of activities based on the rework probabilities and impacts. Given a maximum number of potential reworks for each activity, various rework scenarios can be defined, each with a lower-triangular DSM. By looking across all such scenarios, we can infer an additional characteristic of each activity, its probability of occurrence. This probability guides the activity's selection into a chromosome. A single GA is then performed with variable-length chromosomes. The preparatory procedure proceeds as follows:

1. Enlarge the activity set. Define the set of all possible activities to be performed, both original and reworked activities. For example, if activity 1 in project A (A1) can be reworked up to two times, then two new activities are defined and denoted as A1f and A1ff.
2. Determine activity properties. Original activities maintain their properties, while reworked activities are defined by their durations, resource needs, predecessors, and probability of occurrence. If an activity is being reworked for the first time, its predecessor is the original activity that initiated the feedback cycle; in its *r*th rework, its predecessors are the *r*th rework level activities of the initial predecessors of its original associated activity. As a demonstration, consider the example in Fig. 6. This project (as depicted by its DSM) consists of six original activities. When the activity set is enlarged to include reworked activities 2f, 3f, and 2ff, then this results in a set of nine activities. For original activities, the probability of occurrence is 1, while for reworked activities their probability of occurrence depends on the feedback probability noted in the DSM. For example, the probability of occurrence for activity 2f is 0.5, 3f is 0.1 (0.5×0.2), and 2ff is 0.134 (the union of two paths: $0.5 \times 0.2 \times 0.7 \times 0.5 + 0.5^3$). The durations of reworked activities are calculated according to the following rule: Activity duration = initial activity duration × rework impact ×

[3] Note that it is unnecessary to take the feedback probabilities into account when regrouping the data. These data can be treated equiprobably because the sample is assumed to show all possible scenarios. Each duration already has a frequency of occurrence, which means when analyzing the data we do not need to consider the probability of existence for any chromosome present in the final generation.

Task	Task Detail	Predecessors
1	Original work	-
2	Original work	1
3	Original work	2
4	Original work	2 3
5	Original work	4 all rework cycles initiated by task 4
6	Original work	5
2f	1st order rework	4
3f	2nd order rework	2f 3
2ff	1st order rework second time	4f 2f

	1	2	3	4	5	6
1						
2	0.1			0.5		
3		0.2				
4		0.5	0.7			
5				0.1		
6					0.1	

Fig. 6. Enlarged activity set for the original 6-activity project.

(1 – learning factor). Resource needs for reworked activities are assumed to be the same as for the original activity (albeit typically over a lesser duration).

3. Select an initial population from the enlarged pool of activities.

After the preparatory procedure, the VLGA proceeds as follows (see Fig. 7):

1. Form an initial population, which is a set of possible combinations of activities. The number of combinations or chromosomes is defined by the population size (N), which must be sufficiently large to properly represent the original RCMPSPwF.
2. Test the initial population for feasibility and make any necessary feasibility adjustments with permutations (similar to step 2 in the SGA process described earlier in §4.2).
3. Calculate the fitness of each chromosome as the reciprocal of its duration.
4. Group chromosomes into subsets according to their lengths. Each group will then have a number of chromosomes called

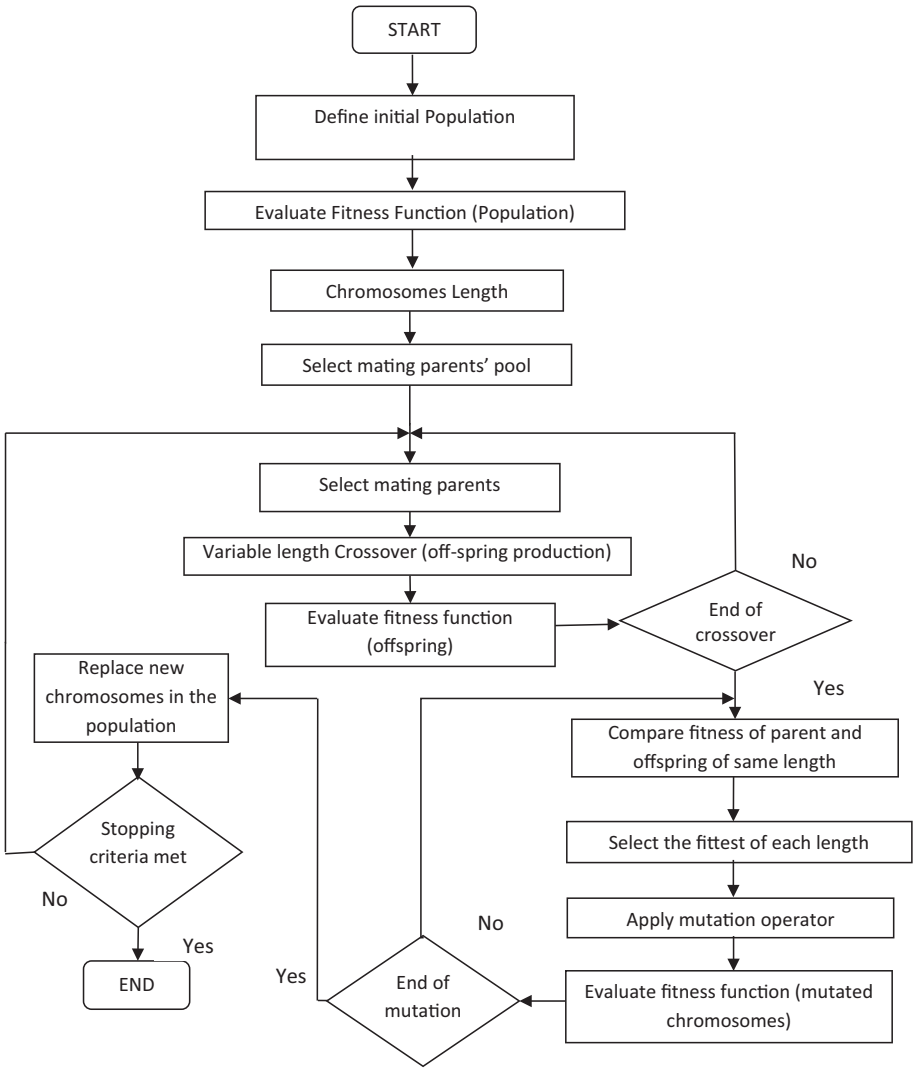


Fig. 7. VLGA flowchart.

	1	2	3	4	5	6
1						
2	0.1			0.5		
3		0.2				
4		0.5	0.7			
5				0.1		0.2
6					0.1	

A population of 12 chromosomes resulting from the DSM:

Chr1:	1	2	3	4	5	6						
Chr2:	1	2	3	4	2f	5	6					
Chr3:	1	2	3	4	2f	3f	5	6				
Chr4:	1	2	3	4	2f	3f	4f	2ff	5	6		
Chr5:	1	2	3	4	5	6	5f					
Chr6:	1	2	3	4	2f	3f	4f	2ff	5	6		
Chr7:	1	2	3	4	2f	3f	4f	5f	5	6		
Chr8:	1	2	3	4	2f	3f	4f	5	6	5f		
Chr9:	1	2	3	4	2f	3f	4f	2ff	3ff	4ff	5	6
Chr10:	1	2	3	4	2f	3f	5	6	5f	6f		
Chr11:	1	2	3	4	2f	3f	5	6	5f	6f	5ff	
Chr12:	1	2	3	4	2f	3f	4f	5	6			

Step 4:

	Group1	Group2	Group3	Group4	Group5	Group6	Group7
Chromosome Length	6	7	8	9	10	11	12
Chromosomes in length groups	Chr.1	Chr.2 Chr.5	Chr.3	Chr.12	Chr.4 Chr.6 Chr.7 Chr.8 Chr.10	Chr.11	Chr.9

Group	Probability of selection
Group 1	= 1/12 = 0.0833
Group 2	= 2/12 = 0.1667
Group 3	= 1/12 = 0.0833
Group 4	= 1/12 = 0.0833
Group 5	= 5/12 = 0.4167
Group 6	= 1/12 = 0.0833
Group 7	= 1/12 = 0.0833

Step 5:

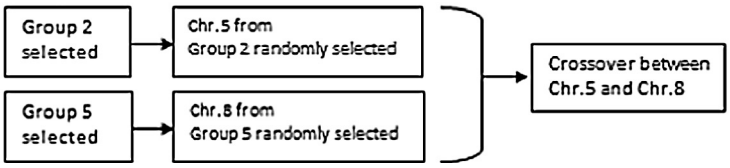


Fig. 8. Example of steps 4–5 of VLGA.

its length counter, τ, and a probability of selection defined as: $\frac{number\ of\ chromosomes\ in\ group}{number\ of\ total\ chromosomes\ in\ the\ population}$. The example in Fig. 8 illustrates this step. The population contains chromosomes of seven different lengths, which is why they are distributed into seven different groups. Chromosomes 2 and 5 are placed in same group since both have length 7.

5. Select mating parents: two random chromosomes are sampled from a random group of a particular length. This maintains the distinct groups and chromosome lengths throughout the generations; otherwise, fitness-based selection alone would merely select the shortest chromosomes (i.e., scenarios with little rework) in successive generations.

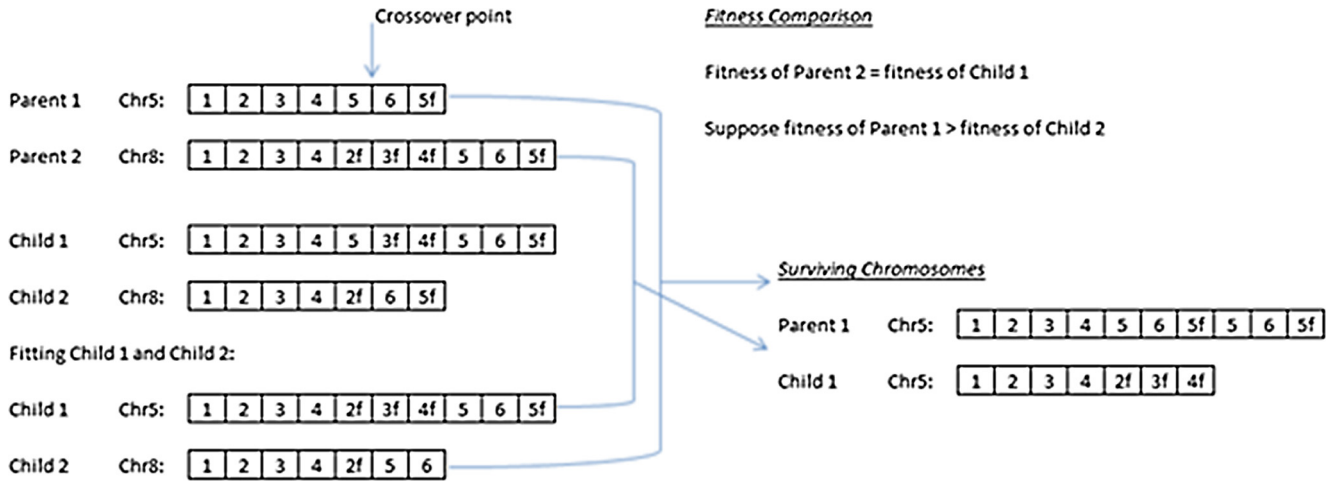


Fig. 9. Example of steps 6–8 of VLGA.

Table 1
Comparing the SGA, SaGA, and VLGA.

	SGA	SaGA	VLGA
Application & Concept	GA is applied to a single RCMPSP problem containing a fixed number of activities	GA is applied to a sample of separate problems (RCMPSPwF is broken down to several (S) RCMPSP without feedback)	Defines additional, reworked activities and extends the portfolio DSM with new precedence relationships
Initial Population	Undergoes GA process	An initial sample of projects is first produced, and project initialization happens separately for each to apply the GA to each project	Undergoes GA process
Feasibility of Individuals	Population may contain both feasible and infeasible individuals	Made feasible by IFC	Made feasible by IFC
Feasibility of offspring	May be infeasible	Made feasible after crossover and mutation	Made feasible after crossover and mutation
Selection for crossover	Direct selection of individuals	Direct selection of individuals	Occurs in two stages: selection of chromosome length and selection of chromosome itself
Crossover	Only between same length chromosomes	Only between same length chromosomes	Between different length chromosomes
Surviving Chromosomes	Children only	Children only	Fittest parent and fittest child
Chromosomes' length in population	Equal	Equal	Variable
Tasks occurrence	Definite	Definite	Probabilistic
End Results	Last generation is considered	Most fit chromosome from the last generation of each sample is considered only, and those of all samples sum up to present the optimal generation of the initial iterative problem	Last generation is considered

Fig. 8 shows how two groups (2 and 5) are randomly selected and a chromosome (5 and 8) is randomly selected from each for mating.

6. Crossover (one- or two-point) may occur between two chromosomes of different lengths. For example, Fig. 9 shows crossover between chromosomes 5 and 8.
7. Each new (child) chromosome is potentially mutated. For example, in Fig. 9, Child 1 and Child 2 resulting from the mating are not feasible.
8. Resulting chromosomes are checked for feasibility. To maintain the presence of groups of all lengths and the survival of the fittest within each such group, the resulting children are made feasible (via the IFC). The child and the parent of the same length are then compared, and the one with highest fitness survives into the next generation. Note that in some cases the parent could survive instead of the child. In the example in Fig. 9, Child 1 and Child 2 are made feasible. Then the child is compared with the parent of same length and the fittest survives.
9. Repeat steps 5–8 until the new generation is produced and the defined population size (N) is reached.
10. Repeat steps 5–9 for each new generation until finishing the Gth generation, which will have a set of optimal or near-optimal durations and project schedules of this problem, with its chromosomes being of different lengths and resembling different possible scenarios of the whole problem.

A large pool containing all of the possible activities is defined. From this pool, an initial population is generated. This population will contain chromosomes of different lengths. The GA is then applied to this population, and crossover occurs between different length chromosomes. Hence, all lengths (rework scenarios) are maintained throughout the generations, and the final generation will contain optimal or near-optimal solutions of different chromosome lengths.

This approach is compact in the number of generations G and execution of GA processes, but huge in population size (N). As the chromosomes are grouped by their length and not by the set

of activities in each scenario, the selection does not consider the activities, only the length of the chromosome. Also, the possible lengths of chromosomes is determined through the initial population, which means if the initialization misses any length of a possible chromosome or scenario, then this length will not show up in any later stage of the GA process. Finally, VLGA does not treat activity durations as variable throughout the entire algorithm. It does that in initiating the initial pool of activities only; but these durations will be fixed throughout the rest of the algorithm. Relative to the SaGA, the VLGA takes longer to generate the initial population but less time to process each generation. The VLGA ends by considering the final generation as a set of optimal or near-optimal solutions for the possible scenarios or schedules of the problem/portfolio. The chromosomes of the final generation and their durations are used to form a distribution of duration outcomes. Table 1 reviews the differences between the SGA, SaGA, and VLGA.

5. Computational results and insights

The previous sections introduced two new GA approaches for use in resource-constrained single and multi-project scheduling problems with feedback. In this section, we present results from validation tests against benchmark problems and comparisons with results from priority rule (PR) heuristics.

5.1. Validation with standard benchmarks

The first validation testing for the proposed GA algorithms was performed using benchmark test problems constructed by the project generator ProGen (Kolisch & Sprecher, 1997; Kolisch, Sprecher, & Drexl, 1995).[4] We used problem instances with 30, 60, 90 and 120 non-dummy activities. These twelve benchmark projects are: (j301_1, j302_5, j3048_10), (j601_1, j602_5, j6048_10), (j901_1, j902_5, j9048_10), and (j1201_1, j1202_5, j12048_10). Since these projects are not iterative, the proposed GAs become identical. The results indicated 0.00% deviation from optimality in all twelve problems and a computational execution time of 19, 46, 88, and 160 s for 30-, 60-, 90-, and 120-activity projects, respectively. These results confirm that our model is valid for single-project instances without iteration.[5]

5.2. RCMPSPwF data

We also tested against a set of iterative, multi-project portfolios from Browning and Yassine (2010a, 2010b, 2016). This set contains three projects per problem, 20 activities per project, 15 feedbacks per project, and four types of resources per activity. The activities in each project are randomly assigned a duration, *dil* (an integer in [1,9]), and requirements for each of K types of resources, *rilk* (an integer in [1,9]). The project portfolios in this set were also controlled through four main factors varied across three levels each: resource distribution or NARLF $(-2,0,2)$, level of resource constraint or MAUF $(0.7,1.1,1.5)$, rework probability $(0,0.1,0.25)$, and network density (complexity) level (HHH, HLL, LLL), where HHH stands for a problem/portfolio composed of three high-density networks, LLL for three low-density networks, and with HLL a mixture with one high-density and two low-density networks.[6] Three replications were considered for each of the $3^4 = 81$ combinations of factor settings, resulting in 243 problems. Browning and Yassine (2016) used *Projects' Average Percent Delay* (O1) and *Portfolio's Percent Delay* (O2) as the objective functions to perform their analyses of PR performance on these problems.[7]

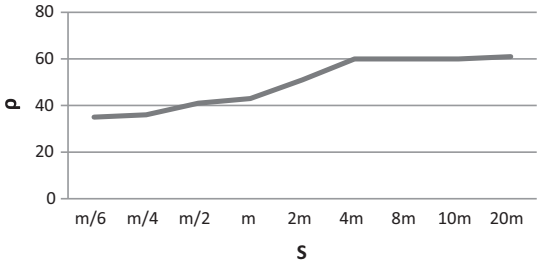


Fig. 10. Range of number of activities (ρ) as a function of Sample Size (S).

5.3. Model calibration via sensitivity analyses of GA factors and problem characteristics

In this section, we use sensitivity analyses to calibrate the GA parameters based on the problem characteristics described in Section 5.2. For calibration purposes, only the significantly differentiated values for each of the main factors were considered: NARLF $(-2,2)$, MAUF $(0.7,1.5)$, rework probabilities $(0,0.25)$, and network density (HHH, LLL). To compare our results with those of Browning and Yassine (2016), we use their two objective functions O1 and O2 in our analyses in this section (instead of project or portfolio duration used earlier in this paper), and we only compare with the relevant subset of their results.

In particular, we use this data set to study the effect of the GA parameters crossover factor (C), mutation factor (X), number of generations (G), and population size (N) on solution quality. Additionally, we study the interaction between the GA parameters and problem settings (i.e., NARLF, MAUF, rework probability (P), and network density (D)) on solution quality and model performance. We use these results to calibrate the SaGA and VLGA parameters for use in comparing their performance to that of the PRs in the Browning and Yassine (2016) study. To study the effect of the network generators on the choice of the GA parameters, both project and problem (portfolio) characteristics and the GA parameters are varied, and the results (O1 and O2) are presented in main effect plots.

In non-iterative projects, the number of activities is fixed, but this is not the case for our formulations of the iterative projects. The existence of different scenarios with different numbers of activities in iterative projects makes it important to define the sample size (S) in the SaGA such that it adequately represents the portfolio. For example, considering $P = 0.25$, $D =$ HHH, NARLF = 2, MAUF = 1.5, and $S = m$ showed chromosomes lengths from 75 to 106 activities (range difference, $\rho = 106 - 75 = 31$), while $S = 2m$ showed lengths from 71 to 122 activities ($\rho = 51$). Increasing S increased the range of chromosome lengths and thus gave a wider presentation of the problem/portfolio. We use ρ as a choice criterion for S as shown in Fig. 10. This plot results from determining the shortest and longest chromosomes generated according to S. Fig. 10 shows that it is best to choose S equal to $4m$, because beyond $4m$ no significant change in ρ occurs, thus causing an unnecessary increase in GA execution time.

Fig. 11 shows the interaction between both number of generations (G) and population size (N) for O1 and O2 based on the network density (D) of the portfolios. Both SaGA and VLGA are applied to the data set considering different combinations of G

[4] These instances are available in the project scheduling problem library PSPLIB from the University of Kiel (http://www.mpsplib.com).

[5] These examples (and all others throughout the paper) were handled using an Intel® processor Core™ i7-3770S CPU @ 3.1 GHz (8 CPUs).

[6] Check Appendix B for a more detailed description of this data set and definitions of the various problem characteristics used in the analysis of this section.

[7] Check Appendix B for a thorough explanation of O1 and O2.

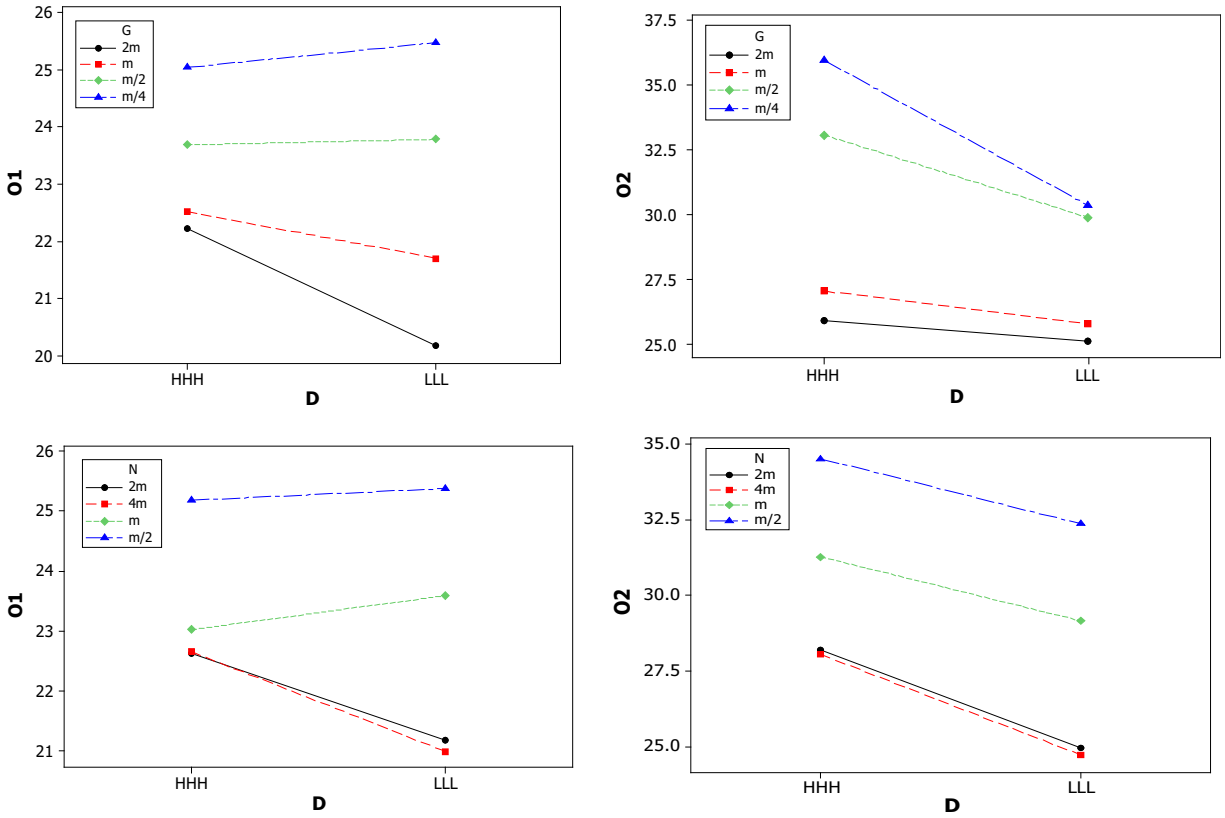


Fig. 11. Number of generations (G) and population size (N) factors' sensitivity to network density (D) (using SaGA with $C = 0.7$, $X = 0.02$).

($m/2$, m, $2m$, $4m$) and N ($m/4$, $m/2$, m, $2m$). As G increases, O1 and O2 decrease, although this decrease is less significant beyond $G = m$. Similarly for N, the significant decrease is for m and $2m$. The impact of both GA factors is larger in less dense projects because these projects have fewer precedence constraints, which increases the number of feasible solutions.

Similar analysis and plots can be obtained for the sensitivity of G and N to MAUF and NARLF, respectively. These plots are shown in Appendix A. The plots show that higher levels of resource constraints (MAUF = 1.5) increase the significance of the impact of both G and N on O1 (project lateness). This effect is even greater on O2 (portfolio lateness). Additionally, for MAUF = 0.7, no sound enhancement is achieved by increasing G and N. The presence of resource constraints increases the number of possible networks as well as possible starting times for an activity. The interaction plots for NARLF show that the effect of resource loading (NARLF) is less pronounced than resource utilization (MAUF) as confirmed by the ANOVA analysis in Appendix C.

Fig. 12 shows the interaction of G and N with rework level (P) for both O1 and O2 using the SaGA. Although not shown in the paper, a similar figure was obtained when running the same analysis using the VLGA. As P increases, the project network becomes larger.[8] The increased number of activities due to iteration does not significantly impact the behavior of G or N (at different levels of P). That is, there is no significant interaction between G (or N) with different levels of P. The effect of rework is tackled by the choice of S in the SaGA and by the initial population size (M) in the VLGA. Table 2 summarizes the calibrated GA parameters used to run the test examples in the next section. Additionally, we have tested the impact of using tournament selection (instead of roulette selection) and using two-point crossover (instead of one-point). There are no significant statistical differences among the results for O1 or O2 means, and accordingly either selection approach (roulette or tournament) may be used in the GA models, as well as any of the one- or two-point crossovers.

5.4. Set up and computational results

Browning and Yassine (2016) compiled a set of 31 PRs, some which have been successful in an acyclical, single-project environment, some from existing literature developed for the acyclical RCMPSP, and others specifically developed for iterative projects. To test the performance of the GAs with respect to these PRs, the data set described in §5.2 is used. The PR solutions are taken from the relevant subset of solutions in Browning and Yassine (2016).

5.4.1. Results for O1

Starting with O1, Figs. 13 and 14 show an one-way analysis of means (ANOM)—without and with iteration, respectively—for all 31 PRs and the 2 GA approaches (averaging over all other factors), showing 95% confidence intervals. First, we note that both proposed GA approaches performed almost identically.

Also, without iteration, the GAs resulted in smaller overall means of delay, but their outperformance of the PRs was not statistically significant. Finally, the GAs significantly outperform the PRs when iteration exists.[9] The main underlying reason for the superiority of the GAs is their ability to predict the future by looking ahead at the full implications of any particular rework loop; this gives it an advantage in allocating resources with foresight.

[8] Much more than merely increasing P, this is really at the heart of the difference between acyclical and cyclical projects (no iteration vs. significant iteration).

[9] The average execution time for the SaGA on all problems with iteration was 70 min (with a standard deviation of 3.6 min). The average execution time for the same set of portfolios using VLGA was 52 min (with a standard deviation of 5.8 min).

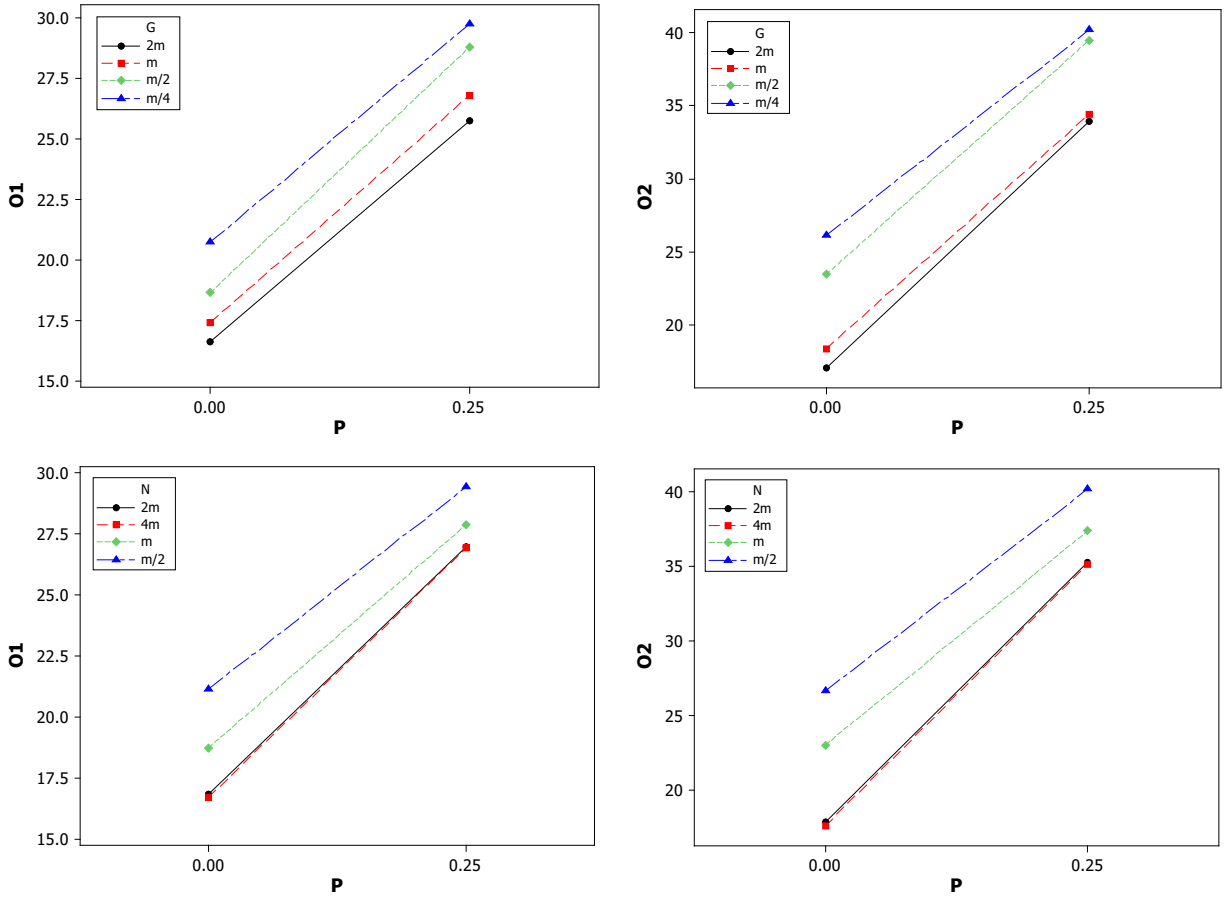


Fig. 12. Number of generations (G) and population size (N) factors' sensitivity to iteration level (using SaGA with $C = 0.7$, $X = 0.02$).

Table 2
GA factors for Calibrated Models.

Factor	GA models	
	SaGA	VLGA
Sample size (S)	$4m$	N/A
No. of Generations (G)	$[m/2, m]$	$[m/2, m]$
Population size (N)	$[m, 2m]$	$4m$ *if m is the No. of all possible activities (original and rework)* $[6m, 10m]$ *(if m is the No. of original activities only)*
Crossover Factor (C)	0.7	0.7
Mutation factor (X)	0.02	0.02

5.4.2. Results for O2

While O1 attends to the effects of delay on the projects individually, O2 accounts for delays that lengthen the overall portfolio. While individual project managers would care more about O1, portfolio managers might focus on O2; however, since O2 is driven by the single longest project in a problem, it is a less sensitive measure than O1 [6].

Figs. 15 and 16 show a one-way ANOM—without and with iteration, respectively—for all 31PRs and the 2 GA approaches (averaging overall other factors), showing a 95% confidence interval. The similarity between the results of O1 and O2 regarding GA performance is due to the fact that the GA approaches the portfolio of projects as one whole project. This makes delays in the single projects highly correlated with that of the whole portfolio. The generation of sequences in a GA based on both resource and precedence constraints without considering start or finish times of activities and projects also contributes to making the results similar.

5.4.3. Summary of results

In the previous section, we identified, confirmed, and discussed several important factors that contribute to project and portfolio delay. To distill these results for managers, we tabulated the results in Table 3 to aid in the selection of GAs instead of PRs for a particular situation. That is, we seek to identify the cases where the GAs outperform PRs. To generate Table 3, we partitioned the dataset into subgroups as shown in Table 3. Then, we performed an ANOM in each subgroup to compare the GAs and PRs in that specific region. For both O1 and O2, the GAs proved superior to the PRs in high-density network problems (HHH), for both moderate (0.1) and high iteration levels (0.25), whenever MAUF is large enough (1.1 and above). As the iteration intensity increases, the GA starts winning over the PRs for high values of NARLF (0 and above) and MAUF (above 1.1). Also, for high values of *NARLF* (2), *MAUF* (1.5), *P* (0.25), and *D* (HHH), GAs proved to be a better approach than PRs. These results are confirmed by the ANOVA in Appendix C, which shows that network density (D), iteration

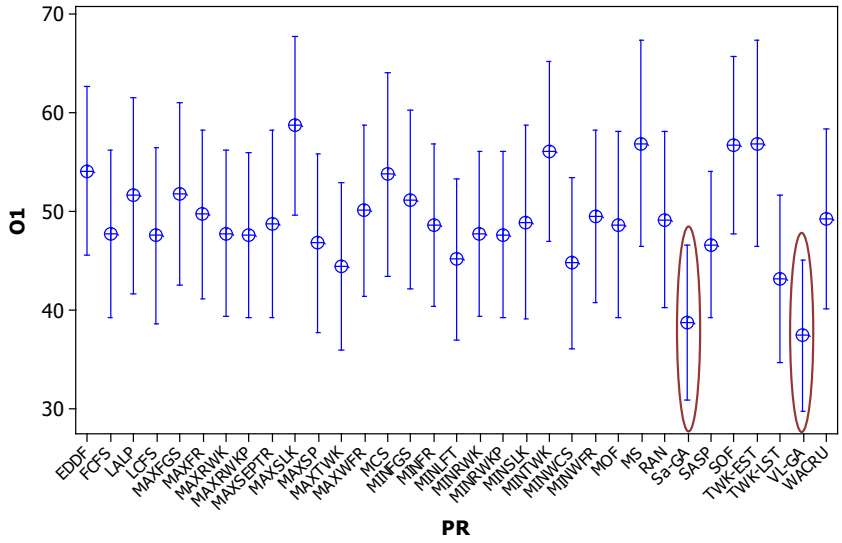


Fig. 13. One-way analysis of means (ANOM) for O1 (no iteration).

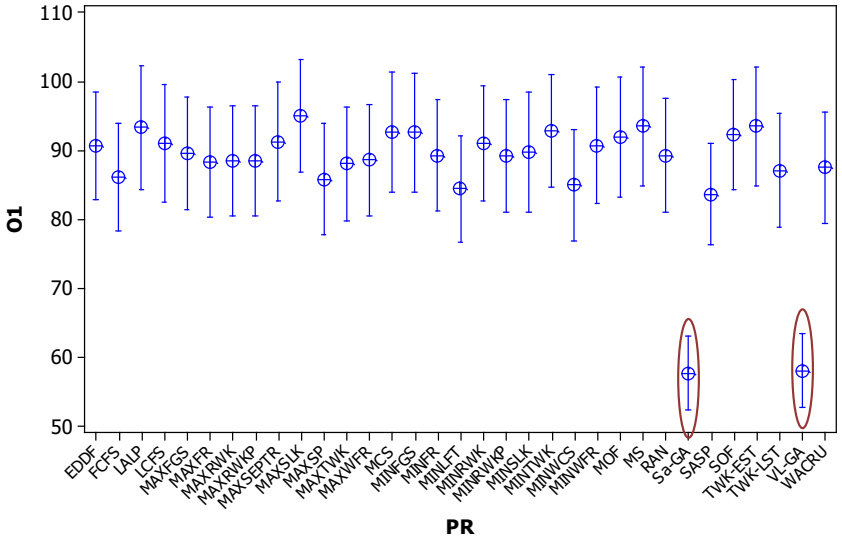


Fig. 14. One-way analysis of means (ANOM) for O1 (with iteration).

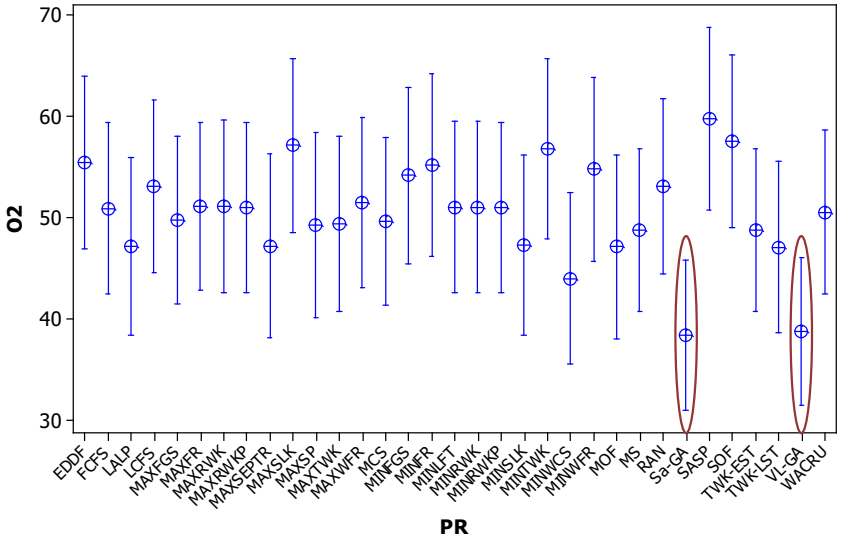


Fig. 15. One-way analysis of means (ANOM) for O2 (no iteration).

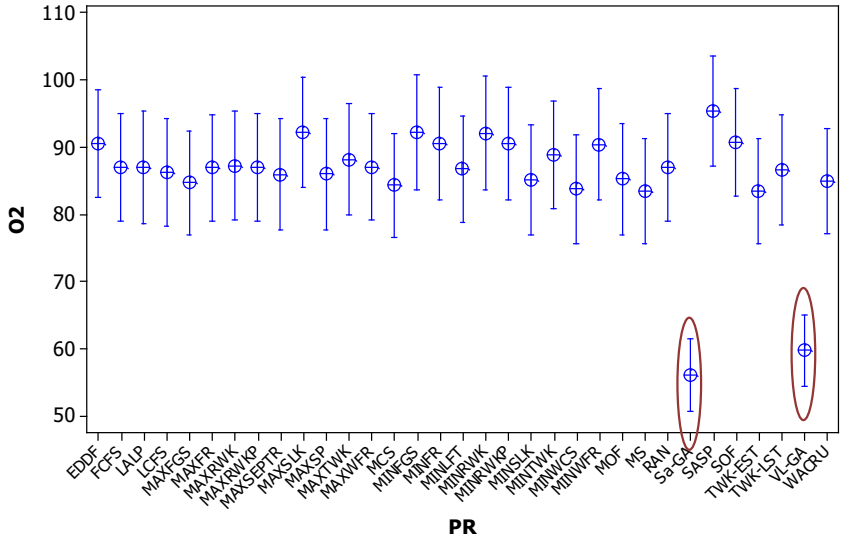


Fig. 16. One-way analysis of means (ANOM) for O2 (iteration only).

Table 3
Summary results for O1 & O2.

ITERATION =0	NARLF=-2			NARLF=0			NARLF=2		
D	HHH	HLL	LLL	HHH	HLL	LLL	HHH	HLL	LLL
MAUF=0.7	PRs			PRs			PRs		
MAUF=1.1									
MAUF=1.5							GAs		

ITERATION =0.1	NARLF=-2			NARLF=0			NARLF=2		
D	HHH	HLL	LLL	HHH	HLL	LLL	HHH	HLL	LLL
MAUF=0.7	PRs			PRs			GAs	PRs	
MAUF=1.1				GAs					
MAUF=1.5									

ITERATION =0.25	NARLF=-2			NARLF=0			NARLF=2		
D	HHH	HLL	LLL	HHH	HLL	LLL	HHH	HLL	LLL
MAUF=0.7	GAs	PRs		GAs	PRs			PRs	
MAUF=1.1							GAs		
MAUF=1.5									

(a) O1

ITERATION=0	NARLF=-2			NARLF=0			NARLF=2		
D	HHH	HLL	LLL	HHH	HLL	LLL	HHH	HLL	LLL
MAUF=0.7	PRs			PRs			PRs		
MAUF=1.1				GAs			GAs		
MAUF=1.5									

ITERATION=0.1	NARLF=-2			NARLF=0			NARLF=2		
D	HHH	HLL	LLL	HHH	HLL	LLL	HHH	HLL	LLL
MAUF=0.7	GAs	PRs		GAs	PRs		GAs	PRs	
MAUF=1.1									
MAUF=1.5									

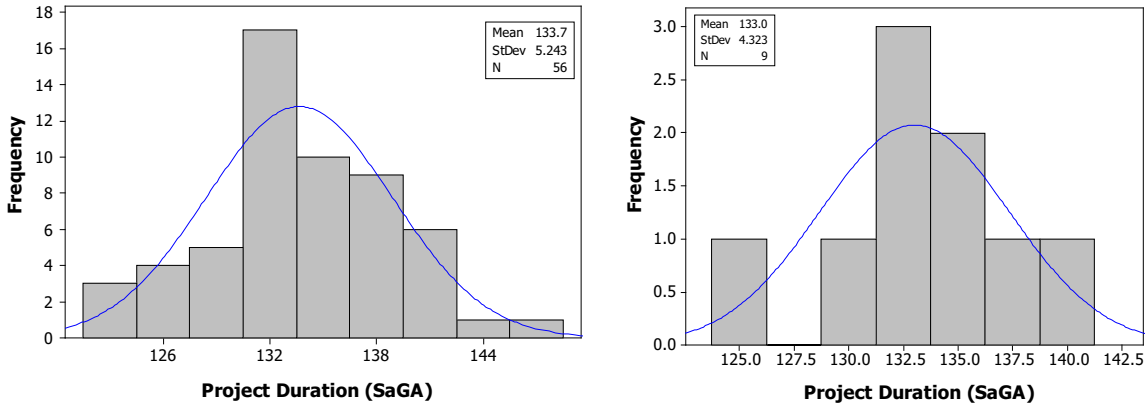
ITERATION=0.25	NARLF=-2			NARLF=0			NARLF=2		
D	HHH	HLL	LLL	HHH	HLL	LLL	HHH	HLL	LLL
MAUF=0.7	GAs	PRs		GAs	PRs		GAs	PRs	
MAUF=1.1									
MAUF=1.5								GAs	

(b) O2

intensity (P), resource distribution (NARLF), resource contention (MAUF), PR/GA type (PRs), and all 2-way interactions are statistically significant (at 99% confidence level). More importantly, the ANOVA shows that 78% of the variance in results can be explained by iteration (P) and resource contention (MAUF), which interestingly highlights the significance of the paper's premise regarding the importance of considering both resource constraints and the iterative potential when making activity scheduling decisions in a PD environment.

To benefit from our results and recommendations as summarized in Table 3, managers must be able to characterize their projects in terms of network density (*D*), amount of resource contention (*MAUF*), resource distribution (*NARLF*), and iteration intensity, *P*. First, regarding *D*, managers can qualitatively estimate whether they are dealing with a high-*D* situation or a low-*D* one without having to precisely obtain a numerical estimate. The approximate density may be determined based on the dominance of sequential or parallel activities, as well as the number of dependencies (precedence constraints) in a project. In addition, the distribution of resources and the qualitative level of resource contention can be ascertained without too much difficulty. However, managers need to define the actual network of activities, in terms of precedence relationships and activity characteristics (DSM), as well as assess the type and quantity of available resources for their portfolios. Having all these data in hand, both the SaGA and VLGA will not only give managers the optimal or near-optimal schedule, but they will also provide them with a statistical distribution and overview for the portfolio of possible sequences to help them account for any lateness due to rework, resource constraints, and precedence constraints.

Using SaGA, the fittest (optimal) chromosome in the final generation for each of the portfolios in the sample is chosen and a distribution is formed from all those chromosomes. Then the solution would be the duration distribution of the portfolio. On the other hand, the chromosomes of the end generation in case of VLGA constitute the solution for the whole portfolio. Considering the distribution from either approach the manager may decide to go with the duration having the highest probability (i.e., frequency in the



(a) Final Duration Distribution using SaGA ($S = 56$, $G = 14$, $N = 28$, $C = 0.7$, and $X = 0.02$)

(b) Modified duration distribution for SaGA

Fig. 17. Distribution for Browning and Eppinger [9] example.

final distribution). This duration may correspond to several schedules or chromosomes and the manger chooses one of them as the project's schedule. However, as the project progresses, iterations unravel and new (i.e., reworked) tasks may occur forcing the manager to deviate from the chosen schedule. In this case, the manager can repeat the analysis to choose a new schedule for the remainder of the project.

To demonstrate the above discussion, we solved the example presented in [9] with the SaGA using parameter settings in accordance with Table 2.[10] The input rework probability and rework impact DSMs for this problem are found in Appendix D. The resultant distribution for duration is shown in Fig. 17(a). Also, the associated chromosomes in the final solutions for the SaGA are shown in Appendix E.[11] A manager may choose a schedule with a duration of around 132, knowing that it is one of the most probable schedules (17% in Fig. 17a). We assume that the chromosome (schedule) chosen is: B1-B2-B3-B5-B4-B6-B7-B8-B5f- B9-B10-B11-B12-B13-B14 (checking Appendix E for all chromosomes of durations close to 132, we chose chromosome number thirteen). If after performing activity 4, activity 3 was reworked (see the rework probability DSM in Appendix D), then the chosen schedule is no longer valid and the manager needs to go with a new schedule. This schedule is also chosen so that it has the most probable duration. Fig. 17(b) shows the modified distribution for the optimal schedules (chromosomes) considering all schedules that start with B1-B2-B3-B4 and activity 3 is reworked (B3f) after activity 4 is performed. For this example there are 9 chromosomes that satisfy the B3f criteria (Appendix E). The manger would select the schedule with highest frequency from this modified distribution, which has a duration of around 132.5. This duration corresponds to a new schedule as follows: B1-B2-B3-B5-B4-B3f-B6-B7-B8-B10-B9-B11-B12-B13-B14 (the seventeenth chromosome in the table of Appendix E).

As mentioned earlier, the VLGA does not treat activity durations as variable throughout the entire algorithm; it only samples a random duration (from a triangular distribution) for each activity in the initial pool. However, these durations are fixed throughout the rest of the VLGA algorithm. On the other hand, in the SaGA, since more than one problem is consolidated at the end, activity durations vary from problem to another. That is, in the SaGA the algorithm generates a new duration each time it chooses an activity. This SaGA property gives the resultant distribution of problem durations more variability. Thus, although the VLGA is faster to execute, we recommend using the SaGA because it provides a richer final distribution.

6. Summary and conclusion

Project and portfolio (multi-project) management are becoming ever-more crucial for the survival of organizations. Relying on their research and development projects, organizations are trying to stay competent and deliver new products and services to assure their success. This competency is based on managing the problems of cost and schedule overrun on projects and finding the best schedule that minimizes project cost, variation of resource profiles, and project duration. Decisions about which activities to do when (based on resource allocations, precedence network details, and iterative activities' integration) have a tremendous effect on project completion times. Yet, many project managers, who often do not have an activity network model to which they might apply more advanced techniques, make resource allocation decisions and project duration estimates based on their own experience or some "rules of thumb" such as PRs.

Iteration is a salient feature of Product Development (PD) projects, where there exists the potential to repeat or rework some of the (upstream) activities based on new information from downstream activities, such as testing and integration activities. This iterative project environment complicates the manager's task of scheduling activities (since some activities may or may not repeat) and assigning resources to them as traditional project management and scheduling techniques (such as CPM or PERT) fail to address iteration. In itself (i.e., without considering iteration), resource constrained project scheduling (RCPS) is a hard problem that is usually approached in practice using established heuristic rules, as these rules are easy to comprehend and implement by managers. However, in iterative, resource constrained PD project scheduling, these rules may be insufficient and misleading due to the existence of iteration.

In this paper, and for the first time, we study the performance of these priority rules (PRs) under iteration and compare them to scheduling results using genetic algorithms (GAs) that provide optimal or close-to-optimal solutions. We find out that in the presence of iteration and under certain project properties, these PRs are insufficient and more involved approaches (such as the proposed GAs) must be considered. We close the paper by providing managers with a useful decision matrix showing when it is best to use the published PRs and when it is best to use the GAs.

[10] Execution time using SaGA was 15 min and using VLGA was 5 min.

[11] Similar results were obtained using the VLGA by plotting the distribution of the chromosomes of the final generation; this analysis proceeds similar to the SaGA analysis described in the rest of this section.

For the study, we generated project portfolios according to a full factorial experiment that included four factors at various levels. Our analysis provides much-needed guidance on the use of GAs in varied project situations and objectives. Accordingly, a decision table is introduced to guide managers in choosing between the best PRs and GA based on MAUF, NARLF, network density (complexity), and iteration level. These results show how different objectives for individual project managers (O1) and portfolio managers (O2) can lead to the preference for different decision rules and thus organizational tensions. The GAs are less favored when there is no iteration, while the chances of GAs winning (statistically) over PRs increases with higher iteration level, MAUF, NARLF, and *D*, considering both O1 and O2.

While several studies regarding project scheduling, PRs, GAs, and related topics exist, it is in some ways astounding that no firmer guidance has appeared for decision makers in the context of iterative projects or portfolios with limited resources, the most realistic situation in contemporary practice. Thus, explaining the conditions under which GAs perform well (or poorly) compared to specific PRs is an important contribution that allows managers to sift through the conflicting results in the literature. However, these results could be immediately applicable in practical situations as long as managers are able to characterize their projects and network details.

Future research could expand our study to combine both GAs and PRs mainly in determining a project or portfolio duration, explore other RCMPSP formulations (such as with preemption, stochastic activity durations, or dynamic project arrivals) in a similarly comprehensive study, or introduce new GA operators that may increase the performance of the GA approach. The results reported can also be used for the development of improved GAs that may perform better than PRs in cases where they failed to do so in this study.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.cie.2017.03.001.

References

Abdelsalam, H. M. E., & Bao, H. P. (2006). A simulation-based optimization framework for product development cycle time reduction. *IEEE Transactions on Engineering Management, 53*(1), 69–85.

Altus, S. S., Kroo, I. M., & Gage, P. J. (1996). A genetic algorithm for scheduling and decomposition of multidisciplinary design problems. *ASME Journal of Mechanical Design, 118*(4), 486–489.

Anderson, S. D., & Tucker, R. L. (1994). Improving project management of design. *Journal of Management in Engineering, 10*(4), 35–44.

Beaujon, G. J., Marin, S. P., & McDonald, G. C. (2001). Balancing and optimizing a portfolio of R&D projects. *Naval Research Logistics, 48*(1), 18–40.

Browning, T. R. (2016). Design structure matrix extensions and innovations: A survey and new opportunities. *IEEE Transactions on Engineering Management, 63*(1), 27–52.

Browning, T. R., & Eppinger, S. D. (2002). Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Transactions on Engineering Management, 49*(4), 428–442.

Browning, T. R., & Yassine, A. A. (2010a). A random generator of resource-constrained multi-project network problems. *Journal of Scheduling, 13*(2), 143–161.

Browning, T. R., & Yassine, A. A. (2010b). Resource-constrained multi-project scheduling: priority rule performance revisited. *International Journal of Production Economics, 126*(2), 212–228.

Browning, T., & Yassine, A. (2016). Managing a portfolio of product development projects under resource constraints. *Decision Sciences, 47*(2), 333–372.

Brucker, P., Drexl, A., Möhring, R. H., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, methods. *European Journal of Operational Research, 112*(1), 3–41.

Demeulemeester, E. L. (2002). *Project scheduling: A research handbook* (102). Springer Science & Business Media.

Demeulemeester, E., & Herroelen, W. (1992). A branch-and-bound procedure for the resource constrained project scheduling problem. *European Journal of Operational Research, 111*(1), 152–174.

Demeulemeester, E., & Herroelen, W. (1996). An efficient optimal solution procedure for resource constrained project scheduling problem. *European Journal of Operational Research, 90*(2), 334–348.

Dou, R., Zong, C., & Li, M. (2016b). An interactive genetic algorithm with the interval arithmetic based on hesitation and its application to achieve customer collaborative product configuration design. *Applied Soft Computing, 38*, 384–394.

Dou, R., Zong, C., & Nan, G. (2016a). Multi-stage interactive genetic algorithm for collaborative product customization. *Knowledge-Based Systems, 92*, 43–54.

Engwall, M., & Jerbrant, A. (2003). The resource allocation syndrome: The prime challenge of multi-project management? *International Journal of Project Management, 21*(6), 403–409.

Eppinger, S. D. (2001). Innovation at the speed of information. *Harvard Business Review, 79*(1), 149–158.

Gendreau, M., & Potvin, J. Y. (2010). *Handbook of metaheuristics* (Vol. 2) New York: Springer.

Gökdağ, H., & Yildiz, A. R. (2012). Structural damage detection using modal parameters and particle swarm optimization. *Materials Testing, 54*(6), 416–420.

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

Gonçalves, J. F., Mendes, J. D. M., & Resende, M. G. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research, 167*(1), 77–95.

Gonçalves, J. F., Mendes, J. D. M., & Resende, M. G. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research, 189*(3), 1171–1190.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics, 45*(7), 733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics, 49*(5), 433–448.

Hartmann, S., & Kolisch, R. (2000). Experimental investigation of state-of-the-art heuristics for the resource-constrained scheduling problem. *European Journal of Operational Research, 127*(2), 394–407.

Hendricks, K. B., & Singhal, V. R. (1997). Delays in new product introductions and the market value of the firm: The consequences of being late to the market. *Management Science, 43*(4), 422–436.

Herm, S. (2013). When things go wrong, don't rely on committed consumers: Effects of delayed product launches on brand trust. *Journal of Product Innovation Management, 30*(1), 70–81.

Herroelen, W. S. (2005). Project scheduling—theory and practice. *Production and Operations Management, 14*(4), 413–432.

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

Hum, S. H., & Sim, H. H. (1996). Time-based competition: Literature review and implications for modelling. *International Journal of Operations & Production Management, 16*(1), 75–90.

Ip, W. H., Li, Y., Man, K. F., & Tang, K. S. (2000). Multi-product planning and scheduling using genetic algorithm approach. *Computers & Industrial Engineering, 38*(2), 283–296.

Kang, C., & Hong, Y. S. (2009). Evaluation of acceleration effect of dynamic sequencing of design process in a multiproject environment. *ASME Journal of Mechanical Design, 131*(2), 021008.

Kerzner, H. R. (2013). *Project management: A systems approach to planning, scheduling, and controlling*. John Wiley & Sons.

Kiani, M., & Yildiz, A. R. (2016). A comparative study of non-traditional methods for vehicle crashworthiness and NVH optimization. *Archives of Computational Methods in Engineering, 23*(4), 723–734.

Kolisch, R., & Hartmann, S. (1998). *Heuristic algorithms for solving the resource constrained project scheduling problem: Classification and computational analysis*. Kluwer, Boston: Handbook on Recent Advances in Project Scheduling.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research, 174*(1), 23–37.

Kolisch, R., & Padman, R. (2001). An integrated survey of project deterministic scheduling. *OMEGA International Journal of Management Science, 29*(3), 249–272.

Kolisch, R., & Sprecher, A. (1997). PSPLIB – A project scheduling problem library. *European Journal of Operational Research, 96*(1), 205–216.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science, 41*(10), 1693–1703.

Kosorukoff, A. (2001). Human-based genetic algorithm. In *IEEE transactions on systems, man, and cybernetics, SMC-2001*. (pp. 3464–3469).

Kurtulus, I., & Davis, E. W. (1982). Multi-project scheduling: Categorization of heuristic rules performance. *Management Science, 28*(2), 161–172.

Lancaster, J., & Cheng, K. (2008). A fitness differential adaptive parameter controlled evolutionary algorithm with application to the design structure matrix. *International Journal of Production Research, 46*(18), 5043–5057.

Lancaster, J., & Ozbayrak, M. (2007). Evolutionary algorithms applied to project scheduling problems—A survey of the state-of-the-art. *International Journal of Production Research, 45*(2), 425–450.

Lenstra, J. K., & Kan, A. H. G. R. (1978). Complexity of scheduling under precedence constraints. *Operations Research, 26*(1), 22–35.

Majava, J., Haapasalo, H., Belt, P., & Mottonen, M. (2013). Product development drivers in literature and practice. *International Journal of Product Development, 18*(6), 512–530.

Mantel, S., Meredith, J., Shafer, S., & Sutton, M. (2007). *Project management in practice*. John Wiley & Sons.

McCulley, C., & Bloebaum, C. L. (1996). A genetic tool for optimal design sequencing in complex engineering systems. *Structural Optimization, 12*(2–3), 186–201.

Meier, C., Yassine, A. A., & Browning, T. R. (2007). Design process sequencing with competent genetic algorithms. *ASME Journal of Mechanical Design, 129*(6), 566–585.

Pennypacker, J. S., & Dye, L. D. (2002). Project portfolio management and managing multiple projects: Two sides of the same coin? In J. S. Pennypacker & L. D. Dye (Eds.), *Managing multiple projects* (pp. 1–10). New York: NY, Marcel Dekker.

PMI (2013). *A guide to the project management body of knowledge: PMBOK guide* (5th ed.). Project Management Institute, Inc..

Rad, P. F., & Levin, G. (2006). *Project portfolio management*. IIL Publishing.

Schwindt, C., & Zimmermann, J. (2014). *Handbook on project management and scheduling* (1 & 2) Cham: Springer International Publishing.

Spinner, M. (1989). *Improving project management skills and techniques*. NJ: Prentice Hall.

Sprecher, A. (1996). *Solving the RCPSP efficiently at modest memory requirements*. Germany: Inst. Für Betriebswirtschaftslehre.

Unger, D. W., & Eppinger, S. D. (2009). Comparing product development processes and managing risk. *International Journal of Product Development, 8*(4), 382–402.

Valls, V., Ballestin, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research, 185*(2), 495–508.

Whitfield, R. I., Duffy, A. H., Coates, G., & Hills, W. (2003). Efficient process optimization. *Concurrent Engineering, 11*(2), 83–92.

Yassine, A. (2004). An introduction to modeling and analyzing complex product development processes using the Design Structure Matrix (DSM) method. *Quaderni di Management (Italian Management Review)* (9), 72–88.

Yassine & Browning (2002). Analyzing multiple product development projects based on information and resource constraints. Unpublished working paper.

Yassine, A., & Braha, D. (2003). Four complex problems in concurrent engineering and the design structure matrix method. *Concurrent Engineering: Research and Applications, 11*(3), 165–176.

Zhuang, M., & Yassine, A. A. (2004). Task scheduling of parallel development projects using genetic algorithms. In *ASME 2004 international design engineering technical conferences and computers and information in engineering conference* (pp. 215–224).