A random keys genetic algorithm for a bicriterion project selection and scheduling problem

Natalia Summerville*

Advanced Analytics and Optimization Services Group,
SAS Institute,
100 SAS Campus Drive, Cary, NC, 27513, USA
Email: Natalia.Summerville@sas.com
*Corresponding author

Reha Uzsoy

Edward P. Fitts Department of Industrial and Systems Engineering,
North Carolina State University (NCSU),
Campus Box 7906, Raleigh, NC 27695-7906, USA
Email: ruzsoy@ncsu.edu

Juan Gaytán

School of Engineering,
Mexico State University,
Cerro de Coatepec, Ciudad Universitaria,
Toluca México, CP 50110, Mexico
Email: jgi@uaemex.mx

Abstract: The project selection and scheduling problem involves the allocation of limited resources to competing projects over time to optimise a given objective function. However, in practical applications, multiple criteria need to be considered, leading us to formulate the problem as a multiple objective combinatorial optimisation (MOCO) model. Activities are subject to precedence constraints, as well as a budget limiting the capital available in each planning period. Interdependencies between projects by which the selection of specific subsets of projects may result in cost savings are also represented. We propose a genetic algorithm incorporating random keys and an efficient decoding procedure into the well-known NSGA-II procedure. The performance of this algorithm is evaluated in extensive computational experiments comparing the approximations of the Pareto-optimal set it obtains to those from NSGA-II.

Keywords: multicriteria optimisation; multiobjective optimisation; genetic algorithms; random keys; project scheduling; metaheuristics; NSGA-II; efficient frontier; evolutionary algorithms.

Reference to this paper should be made as follows: Summerville, N., Uzsoy, R. and Gaytán, J. (2015) 'A random keys genetic algorithm for a bicriterion project selection and scheduling problem', *Int. J. Planning and Scheduling*, Vol. 2, No. 2, pp.110–133.

Biographical notes: Natalia Summerville obtained her PhD in Operations Research from North Carolina State University as well as PhD in Industrial Engineering from Monterrey Tech., Toluca Campus. She was the Director of Undergraduate Studies in Industrial Engineering at Tecnológico de Monterrey, Morelia Campus for four years while teaching in the Department of Industrial Engineering and Business Management. She joined SAS Institute in 2011 and is currently Sr. Operations Research Specialist within the Advanced Analytics and Optimization Services Group. Her research interests include multicriteria optimisation, multiobjective scheduling, price optimisation and revenue management.

Reha Uzsoy obtained his BS in Mathematics from B.S.I.E., as well as MS from Bogazici University and PhD from University of Florida. He is currently a Distinguished Professor in the Edward P. Fitts Department of Industrial and Systems Engineering at North Carolina State University. He is the author of one book, an edited book, and over 70 refereed journal publications.

Juan Gaytán is a Professor of Decision Analysis, Optimisation and Supply Chain at the Graduate School of the Engineering School of Mexico State University (UAEM) in Toluca, Mexico. He received his PhD in Operations Research in 1984. His research interests include: supply chain management, outsourcing decisions, collaborative inventory policies and interactive multicriteria decision techniques. He had provided consulting to companies located in México in simulation, production planning, optimisation and suppliers selection areas. He currently holds the position of Director of Graduate Studies in Decision Analysis at UAEM.

This paper is a revised and expanded version of a paper entitled 'A new genetic algorithm for multicriteria project selection and scheduling' presented at INFORMS Annual Conference, Charlotte, NC, 15 November 2011.

1 Introduction

Researchers from many different areas have recognised the multiple-criteria nature of project selection and scheduling problems (Medaglia et al., 2008; Deb, 2001) whose objectives are conflicting in nature, precluding an easily identifiable, unique optimal solution. Since partial implementation of projects is often not an option, the problem is classified as a multiple objective combinatorial optimisation (MOCO) problem. Most project selection and scheduling problems have been shown to be NP-hard even with a single objective, and are thus likely to be computationally intractable (Lenstra and Rinnooy Kan, 1981). Hence, evolutionary algorithms (EAs) (Zitzler et al., 2000; Fonseca and Fleming, 1995; Jaszkiewicz, 2002; Köksalan, 2008), which attempt to obtain near-optimal solutions in modest computational time, have been widely used. Among these procedures, the non-dominated sorting genetic algorithm (NSGA-II) has been widely used for multicriteria optimisation problems since its introduction in 2001 (Coello Coello, 2009). However, this algorithm was originally designed for unconstrained problems. Deb et al. (2000) suggest ensuring feasibility in the implementations of this algorithm for constrained problems by ensuring that feasible solutions always dominate infeasible ones, regardless of their objective function values. However, due to the large number of solutions generated during the course of any genetic

algorithm (GA), examining each constraint in each solution can result in high CPU times. This issue is important when using GAs for project scheduling problems because conventional crossover or mutation operations can create infeasible solutions.

In this paper, we extend the NSGA-II procedure to the constrained project selection and scheduling problem by using the random keys approach of Bean (1994) to represent feasible project schedule sequences. The combination of the random keys and our greedy decoding method allows any perturbation of a particular solution by mutation or crossover operations to produce feasible offspring. We then apply this algorithm to a set of randomly generated instances of a multicriteria project selection and scheduling problem and perform computational experiments to evaluate the algorithm performance.

2 Previous related work

A MOCO problem involves a number of objective functions to be simultaneously minimised or maximised subject to integer or binary decision variables and a number of constraints that any feasible solution must satisfy. Such problems arise naturally in many applications with a finite, discrete set of feasible solutions. A general formulation of these problems is (Ehrgott, 2005)

$$\min\{f_j(x) : x \in X;\ j = 1,\ldots m\}$$
$$X = \{x : Ax = b,\ x \geq 0 \text{ integer};\ i = 1,\ldots,n\}$$

where f_j denotes the j^{th} objective function and x the vector of decision variables. The matrix A and vector b define the set of linear constraints. A central concept in MOCO problems is the Pareto set, the set of all feasible non-dominated solutions. A solution x^* is defined to be non-dominated, and hence a member of the Pareto set, if $f_{j'}(x^*) < f_{j'}(x)$ for at least one $j' \in \{1, 2,\ldots,m\}$ and $f_j(x^*) \geq f_j(x)$ for $j \neq j'$. In the remainder of this section we review approaches to multiobjective optimisation (MO), emphasising the approximate methods that are the focus of this paper; a complete review of this extensive literature is clearly beyond the scope of this paper.

2.1 Solution methods for MOCO problems

Solution methods for MO are classified based on the involvement of the decision maker (Ehrgott, 2000). In *a priori methods*, all preferences are known at the beginning of the decision making process, and the algorithms seek to generate the complete Pareto set (or a subset of it) on the basis of these preferences. An example of this approach is goal programming (Tamiz et al., 1998; Schniederjans, 1995), where objectives are organised in a pre-specified hierarchy and low-priority objectives are optimised subject to their not degrading the value of a more important one. In *interactive approaches* the decision maker's preferences are introduced during the solution process through a series of computing steps alternating with interaction with the decision maker. An early example of this approach is the interactive branch and bound procedure of Villareal et al. (1979). In *a posteriori approaches*, the set of all Pareto solutions, or an approximation of this set, is generated. This set is then reviewed by the decision-maker who selects a solution based on their preferences.

Due to the NP-completeness of most MOCO problems even with a single objective, much literature focuses on approximation (heuristic) algorithms to address large problem instances. These algorithms aim at generating an approximate Pareto set (or a subset of it) with solutions that either belong to the Pareto set or are sufficiently close to it (Ehrgott, 2000). Algorithms that generate only a subset of the Pareto set must address the issue of diversity among the solutions produced, ensuring that all subregions of the Pareto set are adequately represented in the subset presented to the decision maker.

The most commonly used a posteriori meta-heuristic algorithms have been EAs that use simulated evolution to search for solutions to complex problems (Whitley, 2001). Chromosomes (usually simple data structures) are used to represent solutions that are modified using genetic operators such as crossover and mutation to generate new, and hopefully better, solutions. The ability of EAs to handle complex problems with features such as discontinuities, multimodality, and disconnected feasible spaces has led to their wide use as approximate algorithms for multiobjective programming (Zitzler et al., 2000; Fonseca and Fleming, 1995; Jaszkiewicz, 2002; Köksalan, 2008). The success of EAs in MO is mainly based on their use of a population of solutions, giving them the ability to evolve a diverse population of solutions simultaneously. Veldhuizen (2000) note that EAs are nine times more frequently cited for MO problems than other metaheuristic approaches. However, EAs have difficulty in handling complex constraints, an issue which is the focus of this research. Although EAs have been the most widely used techniques in the literature for MO problems (Crainic and Laporte, 1997), several other metaheuristic approaches such as simulated annealing, tabu search and memetic procedures have been proposed (Ehrgott, 2005; Ehrgott and Gandibleux, 2004), but will not be discussed for the sake of brevity.

The first evolutionary approach to MO was developed by Schaffer (1985) who modified the simple tripartite GA based on selection, crossover and mutation (Goldberg, 1989) by performing independent selection cycles according to each objective. He randomly divided each population into M equal subpopulations, each of which was assigned a fitness metric based on a different objective function. The advantage of this vector-evaluated GA (VEGA) is that is simple and easy to implement. However, because each solution in a VEGA is evaluated with only one objective function, solutions near the optimum of an individual objective function will be preferred, frequently causing VEGA to find only extreme points of the Pareto front (Horn et al., 1994).

To address this weakness of VEGA, Horn et al. (1994) proposed the niched Pareto genetic algorithm (NPGA). This algorithm alters the GA tournament selection where sets of individuals are randomly chosen from the current population and the best subset placed in the next population. This involves adding Pareto domination tournaments and implementing sharing in a non-dominant tournament (i.e., a tie), to determine the winner. The results of the niched Pareto technique for three test instances (two test functions and an application in hydro systems) were encouraging. However, its performance was found to be sensitive to the settings of several parameters such as population size. In particular, it is important to have a large enough population to achieve diversity of solutions and to sample the breadth of the Pareto front.

Zitzler and Thiele (1998) proposed an elitist EA called the strength Pareto EA (SPEA). These authors introduce elitism by explicitly maintaining an external population of non-dominated solutions. A clustering technique is used to improve diversity among the non-dominated solutions obtained. However, this clustering technique has higher computational burden than the crowding sort algorithms in NSGA-II.

Deb et al. (2000) suggested an elitist non-dominated sorting genetic algorithm called NSGA-II. In elitist algorithms, as the name suggests, an elite-preserving operator allows the best solutions of a population to be directly carried over to the next generation. NSGA-II uses an elite-preservation strategy together with a diversity-preserving mechanism implemented through a crowding comparison procedure. The authors introduce the concept of non-domination levels where all solutions are categorised into fronts, the first front being the set of non-dominated solutions, the second front the set of solutions dominated only by solutions in front one, and so on. A more detailed description of this procedure, which forms the point of departure for the work in this paper, is given in Section 4. Among EAs for MO, NSGA-II has been widely used (Coello Coello, 2009). Zitzler et al. (2000) performed experiments on six different test problems for several EAs including NSGA, SPEA and VEGA, and found that NSGA outperformed the other EAs in several quality measures such as the distance to the reference set and the distribution of the non-dominated solutions; computational time was not considered in this experiment. Due to its wide use in the literature and the evidence, albeit limited, of its good performance relative to other EAs, we focus on the NSGA-II algorithm in this paper. However, that the key component of our algorithm, the random keys encoding of Bean (1994), can be implemented in any of the EAs for MOCO problems discussed above.

2.2 Project scheduling and selection problems

The classical project scheduling problem is defined as the allocation of scarce resources to tasks pertaining to the completion of a project over time (Brucker et al., 1999). In contrast, the portfolio optimisation problem involves the constrained allocation of assets towards a subset of available projects, with the most common objective being revenue maximisation (Black and Litterman, 1992). The project selection and scheduling problem considered in this paper shares aspects of both these problems, since it involves the both selection of a subset of projects from an existing set of projects and the scheduling of those projects over time with constrained resources.

A comprehensive multicriteria project selection and scheduling model was solved by Carazo et al. (2010). These authors consider interdependence between projects in the following way: if in period k there are at least m_j and at most M_j scheduled projects from of a set of projects A_j, then there is an increase or decrease in the value of some attribute such as cost. For example, if several projects in a specific set are scheduled there might be a reduction in their total cost. Additional constraints such as available resources, synergy among projects, and limitations on the number of active projects at a given time are also considered, as well as bounds on starting times and precedence restrictions. They adapt the scatter search (Glover, 1994) procedure for MO and compare it to SPEA2 (Zitzler and Thiele, 1998). In their computational experiments SS-PPS outperforms SPEA2 based on the absolute value of the first objective function, disregarding values of the other two to five objective functions being evaluated.

Santhanam and Kyparisis (1995) propose a multiple criteria decision model for multiobjective information technology project selection. Criteria such as corporate priorities, financial benefits and risk are included. Interdependencies are modelled with nonlinear terms in the objective functions and resource constraints. They also consider precedence constraints. They solve the problem using goal programming with preemptive priorities, where the nonlinear terms are linearised with the addition of new variables.

A specific algorithm was developed for a constrained multiobjective project scheduling problem by Viana and Pinho de Sousa (2000), who use Pareto simulated annealing (PSA) (Czyzżak and Jaszkiewicz, 1998) and multiobjective tabu search (MOTS) (Hansen, 1997). They conclude that for this problem, MOTS generally yields better results than PSA in terms of both solution quality and efficiency. To assess the quality of their results, they measure the distance of the solution set to a reference set that ideally should be the Pareto-optimum set, but was defined as the set of the best approximations obtained by running all versions of the algorithms.

Gabriel et al. (2006) present a model for optimal project selection under multiple objectives and random costs. The criteria they consider are project rank, expected project cost, number of managers needed, project risk, impacts on social welfare. They use Monte Carlo simulation to incorporate uncertainty in the data, particularly in the costs. The authors applied this method to a US Governmental agency where 84 projects were initially considered. Another multiobjective model for the selection and timing of public projects with constraints related to starting dates for projects, total number of projects to be selected, precedence constraints, budget and reinvestment was proposed by Medaglia et al. (2008). These authors assign a weight to each criterion and work with a single objective problem. They provide a multiobjective mixed integer linear program that can be incorporated in a user-friendly decision support system.

Jaskowski and Sobotka (2006) developed an EA for a multicriteria construction project scheduling problem with an adapted Tchebycheff function evaluation. They not only scheduled tasks, but also decided which contractors to hire under precedence constraints and renewable constrained resources. The authors solved this as two sub-problems in parallel. The first problem consists of choosing the contractors using an EA; the second problem allocates resources to minimise project duration using a heuristic.

A novel approach to multicriteria optimisation is the interactive analysis of multiple-criteria project scheduling problems developed by Hapke et al. (1998). The problem is characterised by resource constraints, precedence constraints and project performance measures. The first stage of the solution procedure generates an approximation of the non-dominated set using PSA. The second stage is an interactive procedure where after some computations, the decision maker inputs preference information to improve the solutions selected in the next phase.

Rabbani et al. (2010) propose a discrete multiobjective particle swarm optimisation (MOPSO) algorithm for solving the project selection problem with interdependent projects and nonlinear objective functions. Their experiments compare the MOPSO to SPEA II on ten randomly generated test problems. MOPSO generally achieved solutions with higher quality and produces more non-dominated solutions. However, we consider this set of problems to be too small to provide comprehensive insight.

Recent work by Medaglia et al. (2007) proposes a multiobjective EA for a project selection problem where projects can be partially funded. The problem includes multiple stochastic objectives, project interdependencies and linear resource constraints. The key elements of this algorithm are an elitist strategy, parameter-less diversity, a fast stochastic dominance mechanism, and an efficient constraint-handling mechanism that takes advantage of the linearly constrained solution space. Compared to an alternative approach called stochastic parameter space investigation, the results showed that the method is faster and more robust, provides higher quality non-dominated solutions (measured by the average fraction of these solutions belonging to the reference set, which is obtained

by aggregating the solutions from both tested algorithms), and is able to consistently generate a controlled number of solutions that approximate the efficient frontier.

Stummer and Sun (2005) developed a new multiobjective metaheuristic solution procedure for capital investment planning. This is an integer programming project selection model with interdependencies in both objective functions and resource constraints. The authors solve the model with ant colony optimisation, tabu search and variable neighbourhood search and obtain the entire efficient frontier by enumeration. They compare the three algorithms in terms of computation time as well as the percentage of non-dominated solutions found. Computational results on benchmark and randomly generated test problems show that the tabu search procedure outperforms the others if the problem does not have too many objective functions and an excessively large efficient set. The improved Pareto-ant colony optimisation procedure performs better otherwise.

In summary, while a wide variety of EAs have been tested on project selection and scheduling problems, most experimentation has been restricted to very small numbers of problem instances, limiting our ability to draw general conclusions from these studies. In addition, all these approaches suffer in various degrees from the problem of managing constraints to ensure feasibility, resulting in additional computational burden. In the following section we describe the specific problem addressed in this paper, which incorporates both project scheduling and selection.

3 Problem formulation

The initial motivation for this research was a project selection and scheduling problem arising in the tourism sector in Mexico. Each year the State of Michoacán's Ministry of Tourism (MTM), whose responsibility is to promote the development of tourism throughout the state, must allocate its annual budget among a set of projects considered for implementation throughout the upcoming year in order to achieve several objectives. This process involves not only the selection of projects, but also their timing and scheduling. The Ministry's objective is to select projects that will attract more tourists to Michoacán while improving the quality of life in Michoacán's towns and cities within the available budget.

The problem of interest in this paper thus involves the simultaneous selection and scheduling of a number of projects from a given set of candidate projects linked by precedence constraints under multiple objectives. In addition, synergies are identified between pairs of projects such that if two projects i and j are both selected for execution at the same time, cost reductions may result. The primary constraining resource is financing, with a limited budget available in each period. However, unused funds from a period can be carried forward into subsequent periods. We define the following notation:

Decision variables

x_{it} a binary decision variable that takes the value of 1 if project i is started in period t, and 0 otherwise. We shall denote the matrix of all x_{it} values corresponding to a specific solution by X.

Parameters

T number of time periods considered

n number of projects

m number of objective functions

b_{ijt} the benefit generated by investment in project i during period t for criteria j

u_{ik} the reduction in investment generated by synergy between projects i and k if both are started at the same time

S the set of all pairs (i, j) of interdependent projects

Q_i the set of all projects that must be started before project i

D parameter specifying the target density of the precedence constraint graph

q_{ik} binary parameter taking the value of 1 if project i must precede project k, and 0 otherwise

c_{il} the investment needed for project i during its l^{th} period after the start of its execution

r_w budget available for period w

y_w funds available at the beginning of period w

d_i duration of project i in units of time periods.

An integer programming formulation of this problem is as follows:

$$\max f_i(X) = \sum_{i=1}^{n}\sum_{t=1}^{T} b_{ijt}x_{it}, \quad j = 1,\ldots,m \tag{1}$$

subject to

$$\sum_{t=1}^{\max[w-d_k,1]} x_{kt} - x_{iw} \geq 0, \quad i = 1,\ldots,n;\ k \in Q_i;\ w = 1,\ldots,T \tag{2}$$

$$\sum_{i=1}^{n}\sum_{t=\max[w-d_i+1,1]}^{w} c_{i,w-t+1}x_{it} - \sum_{(i,k)\in S} u_{ik}x_{iw}x_{kw} - y_w + y_{w+1} = r_w, \quad w = 1,\ldots,T \tag{3}$$

$$\sum_{t=1}^{T} x_{it} \leq 1, \quad i = 1,\ldots,n \tag{4}$$

$$\sum_{t=T-d_i+2}^{T} x_{it} = 0, \quad i = 1,\ldots,n \quad x_{it} \in \{0,1\}, \quad \forall i = 1,\ldots,n;\ \forall t = 1,\ldots,T$$
$$x_{it} \text{ binary } \forall i, t;\ y_t \geq 0,\ \forall t \tag{5}$$

Constraint set (2) represents the precedence constraints, while (3) limits the funds available in each period. Synergy between projects is represented by the decrease (or increase) u_{ik} in the total cost for projects i and k when both are selected. The first

summation in (3) represents all the costs that need to be covered for the projects selected to start in the time interval up to and including period w. For instance, the cost of starting two projects at the same time might be less than the sum of their individual costs if the same infrastructure is used. This interdependence is considered without regard to the starting time period of the projects and is represented as a quadratic term in (3). This constraint could be linearised using the approaches discussed in Anstreicher (2003), but this linearisation does not yield any significant advantage for the EA approach we adopt. We assume that unallocated funds from previous periods remain available for future projects. Constraints (4) ensure that a project cannot be started twice. A project started in a particular period does not have to be finished in the same period, but can be continued in the following period. Note that this model involves two different interdependencies between projects, precedence constraints and cost synergies, of which the latter are represented in the set S of interdependent pairs of projects.

The formulation above clearly shows the NP-hard structure of the problem due to the presence of the knapsack-like budget constraints (3). Hence, we follow the literature in using an EA to generate an approximation to the set of Pareto-optimal solutions. The details of our proposed solution algorithm are given in the next section.

4 The PS-NSGA-II algorithm

We first review the NSGA-II algorithm of Deb et al., which has proven successful in addressing a range of MOCO problems (Deb, 2001). However, that procedure is designed primarily for unconstrained problems, and its direct application to the problem we address results in high computational burden. We then use the random keys approach of Bean (1994) to obtain an enhanced solution representation that avoids the need for time-consuming feasibility checks within the GA. We first define the terminology used in the rest of the paper.

A solution X is represented by a binary $n \times T$ binary matrix whose element $x_{ij} = 1$ if project i is selected to begin in time period t and 0 otherwise. Here n denotes the number of projects available for consideration and T the total number of time periods. A solution value f is an m-dimensional vector, where m is the number of objective functions, whose j^{th} entry is obtained by evaluating the solution matrix X for criterion f_j. The solution matrix X is thus a natural candidate for use as a chromosome representing a solution within the GA. However, as we show below, this representation creates difficulties due to its inability to maintain feasibility when subjected to genetic operators such as mutation and crossover.

In the NSGA-II algorithm for unconstrained problems an initial population is generated with P random chromosomes. The non-dominated sorting procedure is then applied to generate different fronts, each of which represents a distinct level of non-domination. This non-dominated sorting procedure involves two entities: a domination count n_i, the number of solutions by which each solution i is dominated, and the set S_i of solutions dominated by solution i. The procedure examines each member of the current population, performs the domination evaluation and updates the domination count. After the procedure is completed, the fronts represent an ascending level of non-domination, with front 1 (with domination count equal to 0) containing all the non-dominated solutions.

As an example, consider the following two solution values for four solutions x_1 through x_4 in a maximisation problem where $f_j(x_k)$ is the solution value for criterion j in solution k:

$$f(x_1) = [4, 5] \quad f(x_2) = [6, 4] \quad f(x_3) = [3, 1] \quad f(x_4) = [8, 9]$$

We see that $f_1(x_4) = 8 \geq f_1(x_k)\ \forall k$ and $f_2(x_4) = 9 \geq f_2(x_k)\ \forall k\ \therefore\ x_4 \succcurlyeq x_1, x_2, x_3$, where $\succcurlyeq$ represents domination. By similar reasoning, we can conclude $x_1 \succcurlyeq x_3$ and $x_2 \succcurlyeq x_3$. Therefore, $x_4 \in Front_1$; $x_1, x_2 \in Front_2$ and $x_3 \in Front_3$.

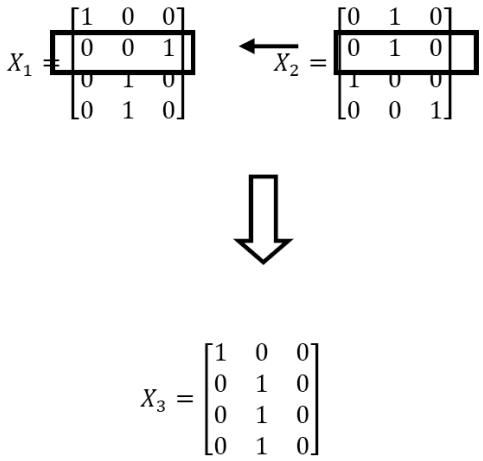
Once the fronts have been identified, a predefined number of chromosomes are selected, giving preference to lower ranked fronts. Solutions from a higher numbered front are not included until all solutions in lower numbered fronts have been included. If there are more than the required number of chromosomes in a front, a crowding sort procedure is performed to select a subset of these based on their distance from each other to ensure coverage of the entire Pareto set. The crossover and mutation operators are then applied, generating the new population, together with the elite solutions from the previous population. If the stopping criterion is satisfied, the algorithm stops; if not, iterations continue. Chromosomes are selected for crossover with a specified probability, and for mutation with a specify probability of mutation. These solutions, together with the elite solutions from the previous population, form the new population. Details of the generic NSGA-II can be found in Deb (2001).

Our implementation of NSGA-II for the problem under study uses the natural chromosome representation derived from the MOCO formulation of the problem given in Section 3. The chromosome takes the form of a matrix X whose entry x_{it} indicates whether project i is started in period t. In order to ensure that no project is started more than once, our crossover operator proceeds by copying entire rows of the matrix as a unit between parent solutions and their offspring. Specifically, we apply one-point crossover by selecting two parents for crossover using a crossover probability of 0.8. We then randomly select a row index k assuming equal probability of selection for each row. The offspring of the two parents is then constructed using rows 1 through k of the first parent and rows $k + 1$ through n of the other. Mutation is implemented by randomly changing the period in which a project is to be started. Once a chromosome is selected for mutation, using a mutation probability of 0.2, a row k is randomly selected, again assuming all rows have equal probability of selection. Suppose that within this row we initially have $x_{kt} = 1$. The mutation operator sets $x_{kt} = 0$ and $x_{gt} = 1$, where g is randomly selected from a discrete uniform distribution on the set $\{1,...,T\}$. The mutation and crossover probabilities were explored in preliminary experiments using a limited number of problem instances, and these values gave satisfactory results.

The difficulty in applying the NSGA-II procedure to our problem arises from the loss of feasibility during application of the mutation and crossover operators, as illustrated in the following example. The natural chromosome representation for our project selection and scheduling problem that would be used by NSGA-II is the X matrix defined above. Suppose we wish to crossover two solutions X_1 and X_2, with four projects and three time periods. Even if the crossover is performed by moving entire rows in order to avoid starting projects more than once, we can obtain the outcome illustrated in Figure 1. Specifically, the budget constraint might be violated by starting several projects in the same period (period 2 in this example), or the precedence constraint requiring that

project 3 has to be completed before project 2 might be violated by starting project 3 ahead of 2.

Figure 1 Example of infeasible crossover



The remedy for this situation suggested by Deb (2001) is to incorporate the feasibility check in the front generating procedure such that an infeasible solution is always dominated by a feasible one. However, over the course of many NSGA-II iterations the algorithm consumes a great deal of time evaluating infeasible solutions. In our experiments with NSGA-II, most of the solutions evaluated in our early experiments were infeasible due to the multiple constraints linking projects and time periods. Our computational results also indicate that this may lead to premature convergence, resulting in a set of solutions that are in fact not actual Pareto solutions.

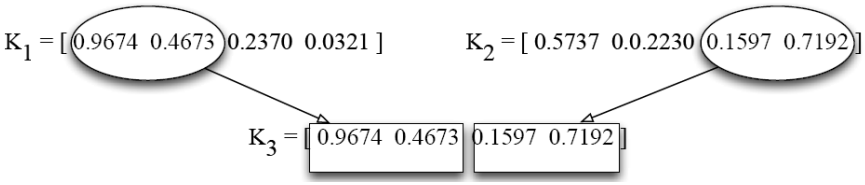
4.1 Random keys

Bean (1994) describes random keys as a "…representation that encode a solution with random numbers. These values are used as sort keys to decode the solution. Random keys eliminate the offspring feasibility problem by using chromosomal encodings that represent solutions in a soft manner. These encodings are interpreted in the objective evaluation routine in a way that avoids the feasibility problem". This approach has been applied successfully to a variety of complex scheduling problems, including the minimisation of total tardiness on parallel machines with sequence-dependent setup times (Norman and Bean, 1999), parallel machine tools (Norman and Bean, 2000), and single and parallel batch processing machines (Malve and Uzsoy, 2007; Wang and Uzsoy, 2002).

Under this representation, the two chromosomes above might be represented by the vectors K_1 = [0.9674, 0.4673, 0.2370, 0.0321] and K_2 = [0.5737, 0.2230. 0.1597, 0.7192] where each element is a random variate uniformly distributed between 0 and 1. Each of these vectors is initially decoded as sort keys R that correspond to a permutation of the projects being considered. In the example above, the vector K_2 is decoded as R_2 = [2, 3, 4, 1] representing a permutation of the projects where project 2 is first, followed by 3, 4 and 1 in that order. When implementing crossover using random keys, two chromosomes are

randomly selected for crossover with a specified crossover-probability. Once the two chromosomes have been scheduled for crossover (K_1 and K_2 in Figure 2), a random number of elements from each chromosome are selected (for this example, the first two first elements in these chromosomes). The crossover is then performed as illustrated in Figure 2: the two elements are taken from K_1 and substituted for the two elements in K_2, generating a new chromosome K_3.

Figure 2 Crossover example using random keys



To implement mutation, a chromosome is randomly selected for the mutation procedure with a mutation probability, a random number of elements are selected and new values for these selected elements are randomly generated from a Uniform[0, 1] distribution.

4.2 Greedy decoding algorithm

The random keys encoding described allows us to search over all permutations of the projects, ensuring that only feasible permutations will be generated. In order to provide feasible solutions to the project selection and scheduling problem we need a decoding mechanism that will map a random keys chromosome into a feasible solution. This is accomplished using a greedy decoding algorithm, which is described below.

The input for the greedy decoding algorithm is the vector K of random keys, decoded into a permutation R of the projects as described above. Given such a permutation of the projects, the heuristic first randomly selects one of the objective functions with respect to which the schedules will be evaluated. The algorithm then considers the projects one by one in the order in which they appear in the *Rank*. Each project, when selected, is tentatively scheduled to begin in the time period that generates the best value of the selected objective function. Once the project has been tentatively scheduled, all constraints are checked to determine whether this assignment is feasible or not. If any constraint is violated, the project is tentatively rescheduled to the time interval that yields the next best value of the randomly chosen objective. If no interval permits a feasible assignment, this project is discarded and the next project in the sequence is selected. The pseudocode is given in Figure 3 using the notation defined in the previous section, with additional notation defined in the figure. The number of operations required to check feasibility is $O(Tn^2)$, where T is the number of periods and n the number of projects.

It is straightforward to see that all solutions generated in this manner from the random keys chromosomes are feasible. This is due to the feasibility evaluation performed in the greedy decoding algorithm; if a particular project is scheduled for a particular time period and the feasibility evaluation fails, then the project is not scheduled for this time period. The procedure is greedy in nature, since a project that is discarded is never reconsidered.

Figure 3 Pseudocode for greedy decoding algorithm

(At iteration *iIter*, given $KEYS_{iIter}$)

For i^* such that $key_{i^*,iIter} > all\ key_{i,iIter}$
  Set $m^* = \text{RANDU}(1, m)$
  While $TIMES \neq \emptyset$
    Select t^*, such that $b_{im^*t^*} > b_{im^*t}\ \forall i, t$
    Set $x_{it^*} = 1$
    Evaluate feasibility
    If not feasible
      Do $sol_{t^*,t^*,iIter} = 0$
      Remove t^* from *TIMES*
    Else break while
    End if
  End while
Remove $key_{i^*,iIter}$ from $KEYS_{iIter}$
End for

$sol_{i,t,iIter}$: element of the solution corresponding to project i, time period t and iteration *iIter*
$key_{i,iIter}$: element of the KEYS set corresponding to project i and iteration *iIter*
$KEYS_{iIter}$: set of all random keys from iteration *iIter*
TIMES: set of time periods

The components of our proposed algorithm, PS-NSGA-II, are now complete. An initial population is created by generating a population of N random chromosomes, where N is a parameter defined by the user. Each random chromosome is represented by a sequence of n values between 0 and 1, using the random keys encoding described above. We then use the greedy decoding algorithm to decode each sequence into a feasible solution matrix X_{ij}. After all chromosomes have been decoded, we perform the non-dominated sorting of Deb (2000) to generate different fronts, and select the number of solutions we need, giving priority to lower rank fronts. If the number of solutions in the last front exceeds the number of solutions we still need to obtain, Deb's crowding sort procedure is performed. Mutation and crossover operators are applied to the chromosomes creating the new population, where the stopping criterion is evaluated. The pseudocode for PS-NSGA-II is given in Figure 4 using the following notation:

- N: population size
- IT: maximum number of iterations to be performed before termination of algorithm
- $FRONT_{iIter,fFront}$: set of solutions in front *fFront* at iteration *iIter*
- $SFRONT_{iIter,fFront}$: sorted subset of $FRONT_{iIter,fFront}$
- $KEYS_{iIter}$: set of all random keys at iteration *iIter*
- $KEYSCROSS_{iIter}$: set of random keys obtained from crossover iteration *iIter*

- $KEYSMUT_{iIter}$: set of random keys obtained from mutation in iteration *iIter*
- SOL_{iIter}: set of solutions at iteration *iIter*.

We now discuss the design and results of the computational experiments used to compare the performance of NSGA-II and PSNSGA-II in the next section.

Figure 4 Pseudocode for the PS-NSGA-II procedure

Set $iterCount = 1$
Generate set of initial random keys $KEYS_0$
Set solutions matrix $SOL_{iterCount} = \emptyset$
While $iterCount \leq IT$
    Set $FRONT_{fCount} = \emptyset$
    Set $numSol = 0$
    Set $fCount = 1$
    Decode $KEYS_{iterCount-1}$ to $SOL_{iterCount}$ using the greedy method for each rank
    Obtain $FRONT_{ij}$ with non-dominated sorting algorithm
    While $numSol < N$
        If $| FRONT_{fCount} | + numSol \leq N$
            Add all solutions from $FRONT_{fCount}$ to $SOL_{iterCount}$
            $numSol = numSol + | FRONT_{fCount} |$
        Else
            Create $SFRONT_{fCount} \subset FRONT_{fCount}$ with crowding sort procedure
            Add solutions from $SFRONT_{fCount}$ to $SOL_{iterCount}$
            $numSol = N$
        End if
    End while
    Obtain $KEYS_{iterCount}$ from $SOL_{iterCount}$
    Perform crossover to $KEYS_{iterCount}$ to get $KEYSCROSS_{iterCount}$
    Perform mutation to $KEYS_{iterCount}$ to get $KEYSMUT_{terCount}$
    $KEYS_{iterCount} = KEYS_{iterCount} \cup KEYSCROSS_{iterCount} \cup KEYSMUT_{iterCount}$
    $iterCount = iterCount + 1$
End while

5 Experimental design

5.1 Generation of random test instances

We explore the performance of our PS-NSGA-II algorithm through computational experiments with randomly generated test instances. The test instances are generated in a manner that allows us to systematically vary the properties of the problem instances that we believe will affect the performance of the algorithms. As mentioned above, for simplicity of analysis we restrict our attention to two objective functions whose

coefficients are generated to ensure that the objective functions are in conflict, requiring effective generation of the Pareto set. Since the emphasis of the paper is on extending the NSGA-II procedure to handle constraints more effectively, we systematically vary the density of the precedence constraints and the strength of the interdependence between projects. The available budget is varied to control the degree to which the budget constraints are binding, which will affect the number of feasible solutions available to the algorithms. Finally, we also vary the size of the problem instances, determined by the number of projects and the number of time periods in the planning horizon, to examine their effect on the computational requirements of the two algorithms compared. The problem instances generated are somewhat smaller than the real-world problem (which had of the order of 50 projects and four objective functions) in order to permit multiple computational experiments examining a wider range of instance and algorithm characteristics than would otherwise be possible. We use uniform distributions (both continuous and discrete) in order to have better control of the values of these parameters, and because the uniform distribution assigns equal probability to extreme values, allowing for the generation of quite diverse problem instances. We shall denote a continuous uniform distribution over the interval $[a, b]$ by $U[a, b]$, and a discrete uniform distribution over the integers between a and b by $DU[a, b]$.

We begin by generating the contribution b_{i1t} of each project i to objective function 1 as a random variate from a $U[0, 100]$ distribution. The contribution b_{i2t} of the project to the second objective is then generated such that $b_{i2t} = 100 - b_{i1t}U[0, 1]$ in order to ensure that the two objective functions are in conflict.

To obtain the savings u_{ik} obtained by starting projects i and k in the same time period, we first randomly generate the number of interdependencies NI from a discrete uniform distribution with parameters

$$NI \sim DU\left(0, round\left(\frac{n}{DU(3, 6)}\right)\right)$$

where n is the total number of projects and *round* denotes the operation of rounding to the nearest integer. The number of interdependencies must be related to the number of projects in the instance. In order to ensure variability in the number of interdependencies in each randomly generated instance, the upper bound of this discrete uniform distribution is also a random variable. We use the pseudocode given in Figure 6 to generate a matrix $U = [u_{ik}]$ containing NI non-zero elements representing the cost savings obtained when projects i and k are started in the same period. We use the numbering of the projects in order to populate only the upper triangle of the savings matrix to save computational time during the main algorithm run.

The q_{ik} representing the precedence relationships are then generated as suggested by Hall and Posner (2001) using a parameter D specifying the target density of the precedence constraint graph, i.e., the probability that a given arc (i, j) exists in the precedence graph. The probability P_{ij} that a given arc (i, j) exists in the precedence graph is then given by

$$P_{ij} = \frac{D(1-D)^{j-i-1}}{1-D\left(1-(1-D)^{j-i-1}\right)}$$

The project costs per time period c_{il} are random variates from a $U(0, 100)$ distribution.

Figure 5 Pseudocode for the generation of the interdependences matrix

For $i = 1$ to NI

$Sav \sim \text{ContinuousUniform}(0, 100)$

$I_1 \sim \text{DiscreteUniform}(n)$

$I_2 \sim \text{DiscreteUniform}(n)$

$Min = \text{minimum}(I_1, I_2)$

$Max = \text{maximum}(I_1, I_2)$

$u_{Min,Max} = Sav$

End for

Budgets r_t for each time period t are randomly generated to ensure that the budget constraint will be sufficiently restrictive that the problem instance will not be trivial to solve, but not so restrictive that there are no feasible solutions. If we have 20 projects, and the cost for a project for a particular period is uniformly distributed in the interval (0, 100), the available budget for period t is a uniform random variate in the interval (0, 100) multiplied by a discrete uniform random variate over the interval (1, 10), ensuring that, on average, the available budget will be enough for 5.5 projects, i.e.,

$$r_t = U(0, 100) * DU\left(1, round(n * BL)\right)$$

where BL represents the tightness of the budget, the smaller the BL, the tighter the budget. The duration d_i of project i is a discrete uniform variate over the interval (1, T) where T is the number of periods in the planning horizon.

We evaluate the performance of NSGA-II and PS-NSGA-II on the basis of both solution quality and CPU time. The discussion of solution quality is clearly complicated by the multiobjective nature of the problem. Due to the particular characteristics of our instances and algorithms, we adopt the approach of creating a reference set from solutions being generated by both algorithms and choosing the Pareto frontier of this set (Hansen, 1998). For a given instance, the solutions from all different runs from both algorithms (for different number of iterations, population, etc.) are collected. The non-dominated sort procedure is performed in order to obtain front 1, which becomes the reference Pareto frontier. For each run, the percentage of solutions that belong to this reference Pareto frontier, denoted by %NDS, is considered as the response variable representing solution quality in this experiment.

It is widely recognised in the literature that the performance of EAs is significantly affected by the choice of values for the parameters used in the algorithm. The population size at each generation is important to ensure that a sufficiently diverse set of solutions can evolve without leading to premature convergence, while maintaining a modest computational burden. The choice of mutation and crossover probabilities also determines the trade-off between intensification of the search in promising areas of the solution space and diversification to ensure all portions of the space are examined.

The termination criterion used is also important to ensure that the algorithm continues to search as long as worthwhile new solutions are being discovered, and that the procedure terminates when there is a low probability of additional worthwhile solutions being found. Both the algorithms tested thus use a stopping criterion composed of two elements:

- *NewNDS*: the minimum number of non-dominated solutions in the new population for the algorithm to continue iterating
- *NumIter*: number of iterations to stop: the minimum number of consecutive iterations during which the algorithm did not find at least *NewNDS*.

Thus, if *NewNDS* = 2 and *NumIter* = 5, the algorithm will stop after five consecutive iterations in which a total of less than 2 new non-dominated solutions at each iteration have been obtained. The tolerance parameter ε is used in the dominance evaluation: solution $\boldsymbol{x}$ is treated as dominating solution $\boldsymbol{y}$ if

- $\dfrac{f_j(\boldsymbol{x}) - f_j(\boldsymbol{y})}{f_j(\boldsymbol{x})} \geq \varepsilon \quad \forall j$
- $\dfrac{f_j(\boldsymbol{x}) - f_j(\boldsymbol{y})}{f_j(\boldsymbol{x})} > \varepsilon$ at least one j.

Thus, a larger value of ε ought to result in earlier termination with a less precise approximation of the Pareto set.

A number of preliminary experiments were conducted on a set of small instances with up to 20 projects and 12 time periods were conducted, varying the values of the algorithm parameters discussed above. Smaller problem instances were used in order to allow identification of the actual Pareto set by explicit enumeration. We initially used a different stopping rule in these experiments, under which the algorithms terminated if no further non-dominated solutions were identified in a specified number of iterations, where the number of iterations was proportional to the population size. Examination of these results indicated that both algorithms were spending a lot of time generating solutions that different only very slightly from previously encountered Pareto solutions, motivating the enhanced stopping criterion described above. The results of these preliminary experiments led to our final experimental design that we discuss in terms of two groups of experimental factors, those related to the algorithms and those related to the problem instances.

5.2 Experimental design

The instance factors that were included in the full experimental designed are listed in Table 1. Parameters *BL* and *D* are used as defined in the previous section. We implemented a full factorial experiment, creating five independent problem instances for each combination of these factors, for a total of 80 instances. The algorithm factors explored in the final experiment are summarised in Table 2. All levels of each algorithm parameter were applied to both algorithms compared.

Table 1 Experimental factors related to problem instances

Instance	*Level 1*	*Level 2*
Number of projects	20	30
Time periods	6	12
Budget level (*BL*)	0.5	1
Interdependence (*D*)	$D \sim U(0, 0.05)$	$D \sim U(0.05, 1)$

Table 2 Experimental factors related to algorithms

Factor	*Level 1*	*Level 2*
Algorithm	NSGA-II	PS-NSGA-II
Population size	20	50
Stopping rule settings (*NewNDS* – *NewIter*)	5 – 5	8 – 3
Tolerance parameter for dominance comparisons	0.01	0.1

With 16 algorithm configurations, five algorithm replicates for each configuration and 80 instances, the full factorial experiment contains 6,400 observations (runs).

The NSGA-II algorithm was implemented as described in Section 2 using the solution matrix X defined in Section 3 as the chromosome. The PS-NSGA-II algorithm was implemented as described in Section 4. Due to the random nature of the GAs each algorithm was run on each instance five times with independent random number seeds. All runs were performed on a Lenovo computer with the following characteristics: Intel® Core™ i7-2620M CPU at 2.7 GHz with 8 GB RAM. The MATLAB version used was version R2013b.

6 Results

Examining the assumptions of classical analysis of variance (ANOVA) related to constant variance and independence of the error terms, we concluded that our data satisfy these assumptions. However, regarding the normal distribution of the error terms, the Anderson test gave a p-value of 0.05, rejecting the null hypothesis that the residuals are normally distributed. Thus statistical analysis using ANOVA is not appropriate. Some transformations, as taking the square root and logarithm, were performed in other to normalise the residuals; the behaviour did improve, but the assumption was still not met. Hence, we analyse the outcome of the experiments using non-parametric tests (Olshen, 1967). In particular, the Kruskal-Wallace test that we use divides the overall set of observations into a number of independent samples such as the observations obtained under different levels of stopping rule, algorithm, and so on, and computes the ranking of each observation within the combined sample. The null hypothesis tested is that the rankings in all samples come from the same population. Extensive discussion of this and other non-parametric tests can be found in Conover (1980).

6.1 Solution quality

The solution quality is measured as the percentage of non-dominated solutions (%NDS). To calculate this percentage, we compute the following quantities:

- $all\_solutions_i$: the total number of distinct solutions obtained from both algorithms for a given instance i
- $opt\_solutions_i$: the total number of Pareto solutions found by both algorithms
- $run\_opt\_solutions_{ij}$: the subset of Pareto solutions for instance i generated by the run j, where a run is defined as a unique combination of algorithm factors.

Then, the $\%NDS_{ij}$ is defined as $run\_opt\_solutions_{ij}$ / $opt\_solutions_i$.

Table 3 Factors influencing % of non-dominated solutions

Factor	*MSE*	*Chi-squared test statistic*	*p-value*
Algorithm	3.54E+09	1,892.42	0.00
Budget	1.12E+08	59.97	0.00
Population	1.03E+08	54.94	0.00
Projects	8.75E+07	46.83	0.00
Tolerance factor for stopping rule	4.88E+07	26.12	0.00
Interdependence	6.54E+06	3.50	0.06
Stopping rule	2.31E+06	1.24	0.27
Algorithm replicate	3.34E+05	0.71	0.95
Time periods	3.05E+04	0.02	0.90

Table 3 summarises the results of the Kruskal-Wallis test (Conover, 1980) across different factors on the percentage of non-dominated solutions %NDS, where the *p*-value indicates the level of statistical significance at which the two levels of each experimental factor can be distinguished. The difference between the two algorithms accounts for the majority of the variance, implying a statistically significant difference between NSGA-II and PS-NSGA-II. The most striking finding in these experiments was that all solutions generated by NSGA-II were dominated by those from PS-NSGA-II. These results suggest that NSGA-II is converging prematurely, before it can reach the solutions identified by PS-NSGA-II. A contributing factor to this behaviour is the large number of infeasible solutions considered by NSGA-II, which cause the stopping criterion to be satisfied prematurely.

As would be expected, the instance related factors of budget and number of projects have significant effect on the %NDS found by each algorithm. This is due to the fact that larger instances require a larger population to maintain a suitable level of diversification and thus prevent premature convergence without examining all areas of the Pareto frontier. A restrictive level of the budget factor impacts the number of feasible solutions, resulting in fewer Pareto solutions overall and making it more difficult for the algorithms to generate a comprehensive representation of the Pareto set in a given CPU time.

The factors that do not have a significant impact on %NDS are also of interest. The interdependence factor describing the level of interaction between projects in the objective functions has a *p*-value of 0.06, which suggests that this factor does in fact have an impact on performance, but that this impact is variable across the overall experiment. The stopping rule does not have a significant impact, which suggests that both setting are achieving essentially the same level of coverage of the search space in terms of Pareto solutions. Further research is necessary to examine whether the set of solutions generated are largely the same, or the two different stopping rules lead to different Pareto frontiers. Finally, the lack of significance of the algorithm replicate factor in encouraging, since it indicates consistent performance by both algorithms on a given instance across independent replications of each algorithm.

6.2 Analysis of computation time

Due to the violation of the linear model assumptions, we will not proceed with regular statistical analysis such as ANOVA, we again use non-parametric tests.

Table 4 summarises the average computation times (in seconds) for each algorithm by the main instance factors. NSGA-II outperforms PS-NSGA-II in computational time, but as explored in previous section, PS-NSGA-II yields significantly better solution quality. An average time of 362 seconds (versus 111 seconds for the NSGA-II) for the larger instances is an acceptable run time, given the improvement in the quality of solutions presented above. The lower CPU time of the conventional NSGA-II does not, in our opinion, represent higher computational efficiency, but rather premature termination of the algorithm due to its inability to detect additional non-dominated solutions.

Table 4 Computational time (seconds) by instance and algorithm

Projects	*Time periods*	*NSGA-II time*	*PS-NSGA-II time*
20	6	32.43	128.94
	12	72.72	210.89
30	6	53.62	208.99
	12	111.30	362.23

Table 5 summarises the influences of each experimental factor on the average CPU time. As would be expected, the most significant factors are those determining the size of the search space (number of projects and number of time periods) and the number of feasible solutions (budget level and project interdependence). The impact of the population size is also significant since this determines the amount of computation taking place at each generation of the algorithms. The lack of impact of the stopping rule once again suggests that the two settings result in approximately the same number of solutions being searched overall. The lack of impact of the algorithm replicate factor once again suggests consistent performance by each algorithm across independent replications.

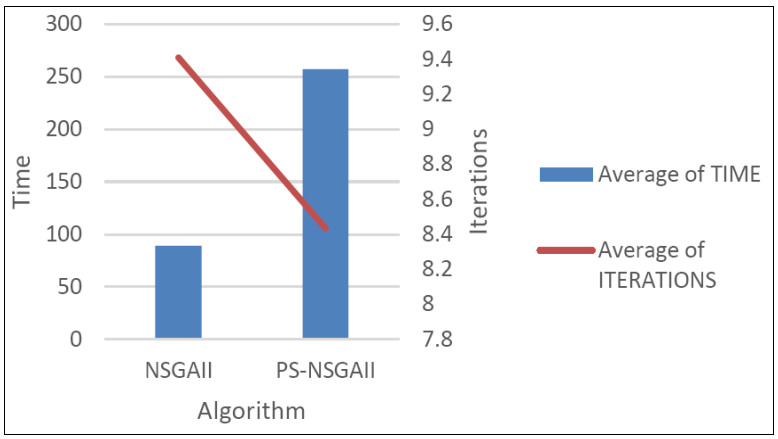
Table 5 Factors influence on CPU time

Factor	*MS*	*Chi-squared*	*p-value*
Population	1.17E+10	3,436.39	0.00
Algorithm	6.04E+09	1,768.32	0.00
Time	1.40E+09	409.61	0.00
Projects	7.31E+08	214.10	0.00
Stopping	5.74E+08	168.18	0.00
Budget	5.63E+07	16.49	0.00
Interdependence	3.40E+07	9.95	0.00
Epsilon	8.86E+06	2.60	0.11
Replicate algorithm	7.73E+04	0.09	1.00

To further explore the differences in CPU time, we analysed the impact of the number of generations on the CPU time for both algorithms, which is shown in Figure 6. While the CPU time per run is higher for NSGA-II, the average of number of iterations required to

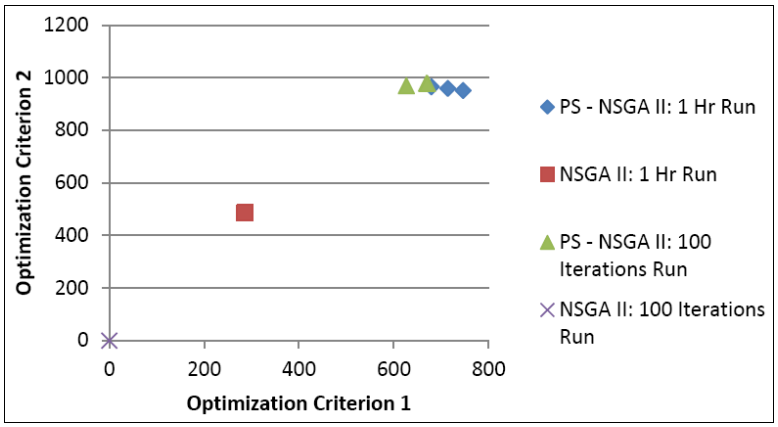
satisfy the stopping criterion is less for NSGA-II, confirming our hypothesis of NSGA-II converging prematurely.

Figure 6 Computational time vs. number of iterations (see online version for colours)



To further examine the behaviour of the CPU time we made runs fixing the CPU time and number of iterations and discarding the stopping rule used in the main experiment. Due to the long CPU times required this was done for a single instance to allow detailed exploration of the behaviour of the two algorithms shown in Figure 7. While this analysis is clearly does not permit any statistical generalisations, we believe it provides some indication of the performance of the algorithms. The results for this instance show that when the number of iterations is fixed at 100, NSGA-II does not return any Pareto solutions at all, and finds only one when allowed a CPU time limit of one hour. In contrast, PS-NSGA-II finds three Pareto solutions after an hour, and two of the same three when run for 100 iterations. This behaviour is not surprising, given that NSGA-II generates many infeasible solutions, while PS-NSGA-II searches over feasible solutions due to the use of random keys. The example also suggests that the number of Pareto solutions for the problem instances in our experiments may be quite small, making them quite difficult test instances.

Figure 7 Long run test results (see online version for colours)



The results summarised above suggest that the proposed PS-NSGA-II algorithm yields substantially better solution quality as measured by %NDS. The computational time of the NSGA-II procedure is substantially less, but this advantage is more than offset by the much lower number of Pareto solutions obtained by the procedure. The key different between the two procedures compared is their treatment of infeasible solutions. The PS-NSGA-II procedure uses the random keys representation to reduce the number of infeasible solutions encountered. By searching over the permutations of projects that are then scheduled in a greedy fashion based on the permutation, a feasible solution can be constructed for each chromosome much more rapidly than is possible by checking all constraints individually for violations. This allows the PS-NSGA-II algorithm to explore more of the solution space for a given population size, resulting in the higher number of Pareto solutions encountered.

7 Conclusions and future directions

Motivated by a real-world application in the tourism sector, we present a comprehensive formulation for the multicriteria project selection and scheduling problem including interdependencies between the projects, in terms of synergies, as well as precedence; allowing the benefits of a project to depend on the time periods in which it is scheduled; and the costs to be time dependent as well. Our proposed PS-NSGA-II algorithm uses the random keys encoding together with a greedy decoding heuristic to ensure that all solutions in a population are feasible, allowing the procedure to generate more Pareto solutions in less time.

A number of interesting directions for future research arise from these experiments. The premature convergence of NSGA-II suggests the exploration of more flexible stopping criteria for both algorithms compared; it may well be the case that longer runs of PS-NSGA-II may yield additional Pareto solutions over those obtained with the current parameter set. A more extensive study of the test instances that reveals the number of feasible solutions would be of interest; we conjecture that many of the test instances have relatively few feasible solutions which makes them particularly challenging for NSGA-II. Finally, the extension of these experiments to a larger number of objective functions will also provide insight into the relative effectiveness of the random keys representation.

Acknowledgements

The research of the first author was supported by a scholarship from the Mexican Foundation for Science and Technology Consejo Nacional de Ciencia y Technologia (CoNACYT).

References

Anstreicher, K.M. (2003) 'Recent advances in the solution of quadratic assignment problems', *Mathematical Programming Series B*, Vol. 97, pp.24–42.

Bean, J.C. (1994) 'Genetic algorithms and random keys for sequencing and optimization', *ORSA Journal on Computing*, Vol. 6, No. 2, pp.154–160.

Black, F. and Litterman, R. (1992) 'Global portfolio optimization', *Financial Analysts Journal*, pp.28–43.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999) 'Resource-constrained project scheduling: notation, classification, models, and methods', *European Journal of Operational Research*, Vol. 112, No. 1, pp.3–41.

Carazo, A.F., Gómez, T., Molina, J., Hernández-Díaz, A.G., Guerrero, F.M. and Caballero, R. (2010) 'Solving a comprehensive model for multiobjective project portfolio selection', *Computers & Operations Research*, Vol. 37, No. 4, pp.630–639.

Coello Coello, C. (2009) 'Evolutionary multi-objective optimization: some current research trends and topics that remain to be explored', *Frontiers of Computer Science in China*, Vol. 3, No. 1, pp.18–30.

Conover, W.J. (1980) *Practical Nonparametric Statistics*, John Wiley, New York.

Crainic, T.G. and Laporte, G. (1997) 'Planning models for freight transportation', *European Journal of Operational Research*, Vol. 97, pp.409–438.

Czyżżak, P. and Jaszkiewicz, A. (1998) 'Pareto simulated annealing – a metaheuristic technique for multiple-objective combinatorial optimization', *Journal of Multi-Criteria Decision Analysis*, Vol. 7, No. 1, pp.34–47.

Deb, K. (2001) *Multi-Objective Optimization Using Evolutionary Algorithms*, Wiley.

Deb, K. et al. (2000) 'A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II', *Parallel Problem Solving from Nature PPSN VI*, pp.849–858.

Ehrgott, M. (2000) 'Approximation algorithms for combinatorial multicriteria optimization problems', *International Transactions in Operational Research*, Vol. 7, No. 1, pp.5–31.

Ehrgott, M. (2005) *Multicriteria Optimization*, 2nd ed., Springer.

Ehrgott, M. and Gandibleux, X. (2004) 'Approximative solution methods for multiobjective combinatorial optimization', *TOP*, Vol. 12, No. 1, pp.1–63.

Fonseca, C.M. and Fleming, P.J. (1995) 'An overview of evolutionary algorithms in multiobjective optimization', *Evolutionary Computation*, Vol. 3, No. 1, pp.1–16.

Gabriel, S.A., Kumar, S., Ordóñez, J. and Nasserian, A. (2006) 'A multiobjective optimization model for project selection with probabilistic considerations', *Socio-Economic Planning Sciences*, Vol. 40, No. 4, pp.297–313.

Glover, F. (1994) 'Genetic algorithms and scatter search: unsuspected potentials', *Statistics and Computing*, Vol. 4, No. 2, pp.131–140.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional.

Hall, N.G. and Posner, M.E. (2001) 'Generating experimental data for computational testing with machine scheduling applications', *Operations Research*, Vol. 49, No. 6, pp.854–865.

Hansen, M.P. (1997) 'Tabu search for multiobjective optimization: MOTS', *Proceedings of the 13th International Conference on Multiple Criteria Decision Making (MCDM'97)*, Cape Town, South Africa, pp.574–586.

Hansen, M.P. (1998) *Metaheuristics For Multiple Objective Combinatorial Optimization*, Working Paper.

Hapke, M., Jaszkiewicz, A. and Slowinski, R. (1998) 'Interactive analysis of multiple-criteria project scheduling problems', *European Journal of Operational Research*, Vol. 107, No. 2, pp.315–324.

Horn, J., Nafpliotis, N. and Goldberg, D.E. (1994) 'A niched Pareto genetic algorithm for multiobjective optimization. evolutionary computation', *IEEE World Congress on Computational Intelligence*, *Proceedings of the First IEEE Conference*, Vol. 1, pp.82–87.

Jaskowski, P. and Sobotka, A. (2006) 'Multicriteria construction project scheduling method using evolutionary algorithm', *Operational Research*, Vol. 6, No. 3, pp.283–297.

Jaszkiewicz, A. (2002) 'Genetic local search for multi-objective combinatorial optimization', *European Journal of Operational Research*, Vol. 137, No. 1, pp.50–71.

Köksalan, M. (2008) 'Multiobjective combinatorial optimization: some approaches', *Journal of Multi-Criteria Decision Analysis*, Vol. 15, Nos. 3–4, pp.69–78.

Lenstra, J.K. and Rinnooy Kan, A.H.G. (1981) 'Complexity of vehicle routing and scheduling problems', *Networks*, Vol. 11, No. 2, pp.221–227.

Malve, S. and Uzsoy, R. (2007) 'A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families', *Computers and Operations Research*, Vol. 34, pp.3016–3028.

Medaglia, A.L., Graves, S.B. and Ringuest, J.L. (2007) 'A multiobjective evolutionary approach for linearly constrained project selection under uncertainty', *European Journal of Operational Research*, Vol. 179, No. 3, pp.869–894.

Medaglia, A.L., Hueth, D., Mendieta, J.C. and Sefair, J.A. (2008) 'A multiobjective model for the selection and timing of public enterprise projects', *Socio-Economic Planning Sciences*, Vol. 42, No. 1, pp.31–45.

Norman, B.A. and Bean, J.C. (1999) 'Genetic algorithm methodology for complex scheduling problems', *Naval Research Logistics*, Vol. 46, No. 2, pp.199–211.

Norman, B.A. and Bean, J.C. (2000) 'Scheduling operations on parallel machine tools', *IIE Transactions*, Vol. 32, No. 5, pp.449–459.

Olshen, R.A. (1967) 'Sign and Wilcoxon tests for linearity', *The Annals of Mathematical Statistics*, Vol. 38, No. 6, pp.1759–1769.

Rabbani, M., Aramoon Bajestani, M. and Baharian Khoshkhou, G. (2010) 'A multi-objective particle swarm optimization for project selection problem', *Expert Systems with Applications*, Vol. 37, No. 1, pp.315–321.

Santhanam, R. and Kyparisis, J. (1995) 'A multiple criteria decision model for information system project selection', *Computers & Operations Research*, Vol. 22, No. 8, pp.807–818.

Schaffer, J.D. (1985) 'Multiple objective optimization with vector evaluated genetic algorithms', *Proceedings of the 1st International Conference on Genetic Algorithms*, L. Erlbaum Associates Inc.

Schniederjans, M.J. (1995) *Goal Programming: Methodology and Applications*, Kluwer Academic Publishers.

Stummer, C. and Sun, M. (2005) 'New multiobjective metaheuristic solution procedures for capital investment planning', *Journal of Heuristics*, Vol. 11, No. 3, pp.183–199.

Tamiz, M., Jones, D. and Romero, C. (1998) 'Goal programming for decision making: an overview of the current state-of-the-art', *European Journal of Operational Research*, Vol. 111, No. 3, pp.569–581.

Veldhuizen, D.A.V. and Lamont, G.B. (2000) 'Multiobjective evolutionary algorithms: analyzing the state-of-the-art', *Evolutionary Computation*, Vol. 8, No. 2, pp.125–147.

Viana, A. and Pinho de Sousa, J. (2000) 'Using metaheuristics in multiobjective resource constrained project scheduling', *European Journal of Operational Research*, Vol. 120, No. 2, pp.359–374.

Villareal, B.K., M.H. and Zionts, S. (1979) 'An interactive branch and bound procedure for multicriteria integer linear programming', *Third Conference on Multiple Criteria Decision Making-Theory and Application*.

Wang, C.S. and Uzsoy, R. (2002) 'A genetic algorithm to minimize maximum lateness on a batch processing machine', *Computers and Operations Research*, Vol. 29, No. 12, pp.1621–1640.

Whitley, D. (2001) 'An overview of evolutionary algorithms: practical issues and common pitfalls', *Information and Software Technology*, Vol. 43, No. 14, pp.817–831.

Zitzler, E. and Thiele, L. (1998) *An Evolutionary Algorithm For Multiobjective Optimization: The Strength Pareto Approach*, Swiss Federal Institute of Technology, TIK-Report, 43.

Zitzler, E., Deb, K. and Thiele, L. (2000) 'Comparison of multiobjective evolutionary algorithms: empirical results', *Evolutionary Computation*, Vol. 8, No. 2, pp.173–195.