A PARAMETER-FREE APPROACH FOR SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS THROUGH BIASED RANDOMIZATION OF EFFICIENT HEURISTICS

DRAGOS IONESCU, ANGEL A. JUAN, JAVIER FAULIN, AND ALBERT FERRER

Abstract. This paper discusses the use of probabilistic or randomized algorithms for solving combinatorial optimization problems. Our approach employs non-uniform probability distributions to add a biased random behavior to classical heuristics so a large set of alternative good solutions can be quickly obtained in a natural way and without complex configuration processes. This procedure is especially useful in problems where properties such as non-smoothness or non-convexity lead to a highly irregular solution space, for which the traditional optimization methods, both of exact and approximate nature, may fail to reach their full potential. The results obtained are promising enough to suggest that randomizing classical heuristics is a powerful method that can be successfully applied in a variety of cases.

1. Introduction

Combinatorial optimization problems have posed numerous challenges to the human mind throughout the past few centuries. From a theoretical perspective, they have a well-structured definition consisting of an objective function that needs to be minimized or maximized and a series of constraints. These problems are important not only at an abstract level though. The main reason for which they have been so actively investigated is the tremendous amount of real-life applications that can be successfully modeled in this way. For example, areas like routing or scheduling contain plentiful hard challenges that can be expressed as a combinatorial optimization problem (see [1]).

A considerable number of methods and techniques that search the solution space and try to find the optimum have been developed. In a few cases, the solution space can be easily explored due to certain properties of the functions involved, such as convexity. For those instances, the problem can be solved efficiently and exactly. However, in most circumstances, the solution space is highly irregular and finding the optimum is in general impossible. An exhaustive method that checks every single point in the solution space would be of very little help

2000 *Mathematics Subject Classification.* 65K05, 90C26, 90C26.

Key words and phrases. Metaheuristics, combinatorial optimization, non-smooth and non-convex optimization.

in these difficult cases, since it would take exponential time. Also, calculus techniques are most often fairly complex and need to take into account the particular features of the problem at hand. Therefore, designing such approaches usually takes a substantial amount of time and methodology has a limited application range. In fact, every method has its drawbacks. We believe that simplicity, efficiency, and the ability to deal with general combinatorial optimization problems under different scenarios are the attributes that can make one approach better or more suitable than another.

The main idea of this paper is that the biased randomization of a classical heuristic can be an efficient way to deal with general combinatorial optimization problems under complex scenarios dominated by non-smooth and non-convex functions and non-convex regions. Basically, our method pertains to the class of nondeterministic or stochastic methods and relies on random sampling. Therefore, on different runs of the algorithm we get different good solutions that depend on which points are randomly sampled. Having a pool of solutions to choose from can be especially useful in real-life problems when the best known solution may be unfeasible due to external constraints.

This article is structured as follows: section 2 review some basic concepts related to convex and smooth optimization problems. Section 3 provides an overview of some general metaheuristics that have been successfully used during the last decades to solve combinatorial optimization problems. Section 4 offers a literature review on the use of probabilistic algorithms (metaheuristics) to solve non-smooth and non-convex optimization problems. In Section 5, our general simulation-based approach is introduced and discussed. Sections 6 and 7 show examples of application of our approach to two classical combinatorial optimization problems, namely, the vehicle routing problem and the flow-shop problem. Finally, the concluding section summarizes the main contributions of this work.

2. Review of convex and smooth problems

Optimization problems can be categorized, from a high-level perspective, as either convex or non-convex. In this section we give an overview of the properties corresponding to these two types, and we point out how the difficulty of finding optimal solutions increases exponentially as you transition from the convex domain to the non-convex one.

In general, convex optimization problems (COPs) have two parts: a series of constraints that are convex regions, and an objective function to be minimized that is convex. The equivalent problem, in which the objective function is concave and the goal is to maximize it, is also often encountered and for the purpose of this paper it will still be considered a member of the convex-like class of problems. COPs are worth studying because they have a wide variety of applications and many problems can be reduced to them. Linear Programming is one well known example, since linear functions are trivially convex. Other examples of COPs

include Quadratic Programming, Geometric Programming, Conic Optimization, Least Squares, etc (Boyd and Vandenberghe, 2004). The main idea, in convex optimization problems, is that every constraint restricts the space of solutions to a certain convex region. By taking the intersection of all these regions we obtain the set of feasible solutions, which is also convex. Due to the nice structure of the solution space, every single local optimum is a global one. This is the key property that permits us to solve COPs exactly and efficiently up to very large sizes. Several algorithms, such as the "Interior Point Method" (Wright 1997), have been developed to find the optimal solution. However, almost none of them can be easily extended to the non-convex case. In non-convex optimization problems (NCOPs) the objective function or even the feasible region are not convex, which results in a far more complex solution space. Now we may have many disjoint regions, and multiple locally optimal points within each of them. As a result, if a traditional local search is applied, there is a high risk of ending in the vicinity of a local optimum that may still be far away from the global optimum. Another drawback is that it can take exponential time in the size of the input to determine that a NCOP is infeasible, that the objective function is unbounded, or that one of the solutions found so far is the actual global optimum (see Aferrer references).

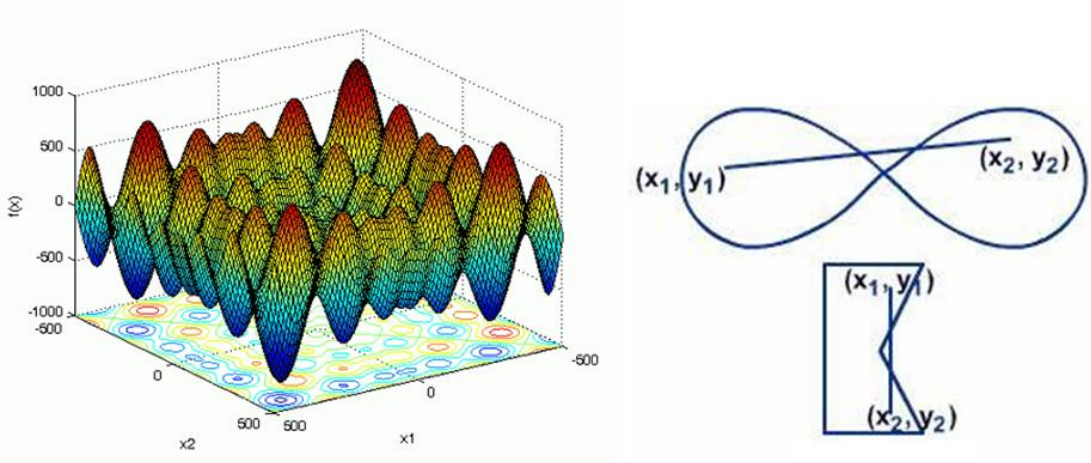

FIGURE 1. Non-convex objective function and feasible regions

A function is smooth if it is differentiable and the derivative is continuous. Therefore, a non-smooth function is one that is missing some of these properties. Non-smooth optimization problems (NSPs) are similar to NCOPs in the sense that they are much more difficult to solve than traditional smooth and convex problems. The function for which a global optimum needs to be computed is now non-smooth and the solution space might contain again multiple disjoint regions and many locally optimal points within each of them. The lack of a nice structure makes the application of traditional mathematical tools, such as

gradient information, very complicated or even impossible in these cases. The computational techniques that can be used to solve this type of problems are often fairly complex and depend on the particular structure of the problem. As a result, developing such techniques is in general time-consuming, and the resulting application range is very limited.

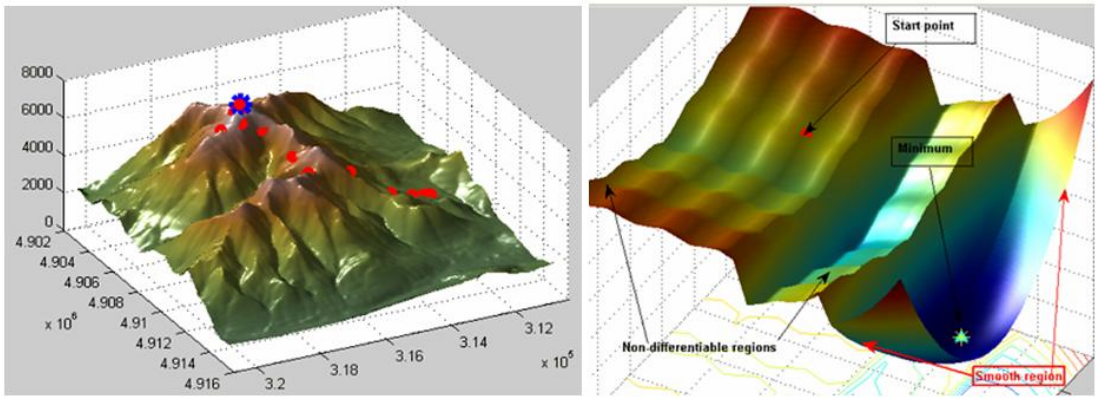

FIGURE 2. Non-smooth functions

3. Overview of general metaheuristics

Metaheuristics are widely used to solve many practical combinatorial optimization problems but due to the variety of techniques and concepts comprised by metaheuristics, there is still no commonly accepted definition of what metaheuristic means. In our opinion, a metaheuristic can be seen as a general algorithmic procedure that can be used to solve many different optimization problems with relatively few modifications to adapt it to a specific problem. Examples of metaheuristics are genetic and evolutionary algorithms, tabu search, simulated annealing and the Greedy Randomized Adaptive Search Procedure (GRASP) among others.

3.1. **Genetic and evolutionary algorithms.** This kind of algorithms represent one potential approach to solve non-smooth and non-convex optimization problems. The main idea here is to mimic the process of natural evolution and make use of genetic procedures such as inheritance, mutation, selection, and crossover. Unlike other approaches, genetic and evolutionary algorithms maintain a population of candidate solutions, rather than just the best solution found so far. The algorithm starts with a set of candidate solutions (the initial population), and generate new ones through random mutation, crossover, or recombination. Then it applies a selection step in which the worst solutions are deleted while the good ones are passed on to the next generation. The entire process is repeated multiple times and gradually better and better solutions are obtained. As in any

other approximation algorithm, the drawback is that there is no way to determine whether a good solution that has been generated is the actual optimum.

TABLE 1. Genetic algorithm

Procedure: Genetic algorithm
begin
P:=GenerateInitialPopulation();
while stop=false **do**
P^1 :=Recombine(P);
P^2 :=Mutate(P^1);
P :=EvaluateAndSelect(P^2);
end while
end

Among the several different evolutionary algorithms proposed over the years, there are three that have attained a relatively high level of popularity: Evolutionary Programming (EP) proposed by Fogel (1962) , Genetic Algorithms introduced by Holland (1975), and Evolutionary Strategies (ES) that are due to Rechenberg (1973).

3.2. **Tabu search.** Tabu Search (TS) is another popular method for solving NSPs. The TS basic ideas have been de-veloped by Glover (1986), and a more detailed description of the method is given in Glover and Laguna (1997). This optimization process is in fact a sophisticated local search technique that employs a memory structure to guide the direction and intensity of the search. The memory structure is most often a tabu list that keeps track of the solutions that have been recently encountered. Also, for each solution, a neighborhood that does not contain any of the elements in the tabu list is defined. Then, as you iterate through the metaheuristic, jumps are made from the current solution to another one in its neighborhood. More exactly, the best or the first found neighbor which offers an improvement is set as the new current solution at each step. The local search finishes when the current solution is better than all its neighbors, i.e. a local optimum has been encountered. But this would be a poor approach in practice since in most cases the local optima are still far away from the global optimum. To avoid being trapped in the vicinity of a local optimum or getting in an infinite loop, TS uses the memory structure mentioned earlier that contains a history of the searches. Because of memory limitations, the entire solutions cannot be saved in the tabu list. Instead, specific solution components, the differences between solutions, or the moves necessary to transition from one solution to another represent the information that is being stored. Saving only partial information about the solutions that have been explored greatly reduces the amount of memory that is necessary for this process, but also introduces some limitations. The problem is that even solutions that have not been visited may

accidently satisfy the properties in the tabu list. As a result, we may overlook possible candidates for a local or global optimum position. To overcome these difficulties, the forbidden moves stored in the tabu list are sometimes permitted if a certain aspiration criteria is satisfied. The given pseudo-code summarizes the entire procedure. In the code, x, y are feasible solutions for the problem, PS(x, i) is the set of possible solutions among which the new current solutions is chosen at iteration i. $S(x)$ is the set of neighbors of x, TL(x, i) and TM(x, i) are the set of tabu moves and the set of moves satisfying at least one aspiration criterion at iteration i. The algorithms stops if a certain threshold time is reached, a maximum number of iterations has been exceeded, or there are no more possible movements.

TABLE 2. Tabu search

Procedure: Tabu search algorithm
begin
x:=GenerateInitialSolution();
InitializeTabuLists();
$i := 0$;
while stop=false **do**
PS$(x, i) := \{y \in S(x) \backslash \text{TL}(x, i) * \text{TM}(x, i)\}$;
x:=ChooseBestOfPS(x, i);
UpdateTabuListAndAspiration Condition();
end while
end

As with Genetic and Evolutionary Algorithms, there is in general no way to test if one of the good solutions is the optimal one.

3.3. **Simulated annealing.** A probabilistic metaheuristic that can be applied to combinatorial optimization problems is Simulated Annealing (SA). SA is based on Metropolis algorithm (1953) used to simulate the evolution of a solid in a heat bath to thermal equilibrium and it was introduced by Kirkpatrick et al. (1983). In this case, we see an example of a method that models a physical process, namely the behavior of atoms under different temperature regimes. It has been observed that heat causes the atoms to deviate from their original configuration and transition to states of higher energy. Then, if a slow cooling process is applied, there is a relatively high chance for the atoms to form a structure with lower internal energy than the original one. By making a simple analogy, such that solutions in the search space correspond to states of the system and the function to be optimized corresponds to the total internal energy, we can utilize these ideas for solving NSPs. There is also a global temperature variable, T, that comes into play as you try to switch from one state to another. In essence, each transition has a probability that depends both on the global temperature

and the energy levels of the current state and the potentially next one. For large T almost any movement is allowed, while for small T only the transitions that improve the current solutions are permitted. By accepting both better and worse solutions, you avoid to be trapped in the vicinity of a local optimum.

The algorithm first generates an initial solution s, and assigns the initial value T_0 to the temperature. . At each iteration a random solution is selected from the neighborhood of the current solution. The solution selected will be the new current solution depending on the energy ($f(s)$ and $f(s_c)$) and temperature T. More exactly, the selected solution becomes the current solution if $f(s_c) \leq f(s)$ or if it gets accepted with a certain probability based on a Boltzmann distribution, $\exp((f(s) - f(s_c))/T)$.

TABLE 3. Simulated annealing

Procedure: Simulated annealing algorithm

```
begin
    s :=GenerateInitialSolution();
    T := T_0;
    while stop=false do
        s_c :=SelectAtRandom();
        if f(s_c) ≤ f(s) then
            s := s_c;
        else
            if exp((f(s) − f(s_c))/T) > random[0, 1) then
                s := s_c;
            end if
            update(T);
        end if
    end while
```

3.4. **The greedy randomized adaptive search procedure.** This metaheuristic was introduced by Feo and Resende (1989). In general, a GRASP algorithm has 2 phases: a constructive phase to build a good solution and a local search phase to further improve the solution. In fact, the construction phase builds a solution whose neighborhood is investigated by the local search procedure. The 2 steps are repeated until a stopping criterion is satisfied and the best solution encountered so far is returned. A greedy function is used to guide the decisions at each intermediate step. In addition to this, a random element is selected at each iteration. Therefore, GRASP pertains to the class of stochastic methods that give different final solutions on different runs of the algorithm. There is also a parameter DELTA that controls the randomization process. It can be either fix or variable, and in the latter case it follows a strategy similar to tit-for-tat. As a result, you are gradually more likely to choose values of DELTA for which

good solutions have been already obtained. During the whole process, the best solution is updated and returned at the end, after a certain number of iterations.

Table 4. GRASP algorithm

```
Procedure: GRASP
begin
    procedure GRASP(maxIt, seed)
        ReadInput();
        for i := 1 to maxIt do
            Solution:=GreedyRandomizeConstruction(seed);
            Solution:=LocalSearchSolution(Solution);
            UpdateSolution(Solution);
        end for;
        return BestSolution;
    end GRASP
```

4. Metaheuristics and combinatorial optimization

In this section we review some papers that employ probabilistic algorithms to solve non-convex or non-smooth combinatorial optimization problems.

Our first example is based on the minimum sum-of-squares clustering problem, that can be formulated as a non-smooth, non-convex optimization problem as noted in Bagirov and Yearwood (2006). The goal of clustering problems is to separate a large set of objects into groups or clusters based on similarity. There are many possible applications, especially in fields such as engineering, information and decision sciences or earth sciences. Information retrieval or image segmentation are some particular cases in which clustering techniques can be applied. Bagirov and Yearwood (2006) also emphasize the fact that a large number of approaches, such as dynamic programming, branch and bound, or K-means algorithms have been tried for the clustering problem. However, the authors point out that most of these methods are efficient only in certain special settings. For example, a dynamic programming approach only works when the number of instances is small, so it cannot be used for real-world problems. Branch and bound methods are effective only if the number of clusters is not too large. The efficiency of K-means algorithms also decreases as the number of clusters is increased. Moreover, alternative techniques such as bilinear programming fail as well when they face non-convex and non-smooth objective functions that have many local optimums. The authors remark that in general better results are generated when metaheuristics, such as the ones presented in the previous section, are used for the clustering problem.

A Tabu Search approach that outperforms the K-means has been proposed by Al-Sultan (1995). However, the algorithm requires 3 parameters, so "an extensive parameter study" has been necessary to find the "best parameter settings". Another simulated annealing approach is due to Selim and Al-Sultan (1991), but it also requires rather complex fine-tuning processes. The issue of optimal routing in communication networks has also received a lot of attention from researchers. The objective here is to find the best path for data transmission in a short amount of time. The routing strategy can greatly affect system performance, so there is a high demand for efficient algorithms. As a result, numerous methods that deal with this challenge have been designed. Ali and Kamoun (1993) develop a neural-network approach for this shortest path problem. A method that combines genetic algorithms with Hopfield networks (see R. Rojas, 1996) has been proposed by Hamdan and El-Hawary (2002). Finally, an approach based on tabu search techniques is taken by Oonsivilai et al. (2009). The main drawbacks for most of these methods were either the inability to efficiently explore the solution space or very long computation times.

The last example we take discusses the problem of localization in wireless sensor networks. In general, a wireless sensor network can be defined as a distributed collection of nodes (sensors) that have limited resources and operate autonomously. The goal is to find or accurately estimate the position of the nodes. This task could be easily accomplished using a Global Positioning System (GPS) for every node, but the costs would be prohibitive. A non-smooth formulation for this optimization problem is give in Bagirov et al. (2010) [could not find this reference]. In addition to this, a lot of research has been done to study both indoor and outdoor localization systems. However, most of the approaches have assumed accurate range measurements, which is unrealistic for RF signal strength measurements. Ramadurai and Sichitiu (2003) show that a probabilistic method can be adopted to deal with range measurements inaccuracy. Their algorithm is "RF based, robust to range measurement inaccuracies and can be tailored to varying environmental conditions". As we have seen in the examples above, probabilistic algorithms have a high potential and often outperform the traditional deterministic methods when we are considering non-smooth or non-convex combinatorial optimization problems.

The approach we propose in this paper is also a randomized algorithm, but it has certain specific characteristics that set it apart from the metaheuristics presented in section 3. For example, the methods in the previous section usually make "jumps" from a solution to another one in its neighborhood, so only small perturbations are allowed. By contrast, our technique can be described as a multi-start process that introduces big changes. Basically, once a relatively good solution has been found in one region of the solution space we restart the search in a totally different region. Moreover, we use a random biased behavior rather than a uniform one, so we manage to add a certain degree of intelligence to our algorithm right from the beginning of the search process. We believe that,

depending on the objective function, this approach may perform better than other metaheuristics designed for solving NSPs and NCOPs. Additionally, our method requires few or no parameters and can find good solutions very fast, qualities that make it more robust than other approaches.

The goal is to point out that these problems can appear in a variety of situations and are usually characterized as having a high degree of difficulty. Even when the original theoretical version of a problem may not pose serious challenges, more constraints are added as you try to adapt it to real-life scenarios so the search for a good solution is no longer trivial. This is why we believe it is important to analyze different approaches for solving NCOPs or NSPs and observe both the advantages and the drawbacks of these methods.

5. Multi-start randomization of classical heuristics (MSCH)

In this section we describe in detail the multi-start randomization methodology for NSPs and we illustrate how it can be successfully applied for two well-known examples, the Vehicle Routing Problem (VRP) and the Permutation Flowshop Sequencing Problem (PFSP). As it has been previously mentioned, our algorithm starts with the solution generated by a classical heuristic and slightly perturbs it by means of a random biased behavior in order to obtain alternative good solutions. We have chosen classical heuristics as our starting point for several reasons. First of all, they usually have excellent performance, are well tested, and have been developed for almost every combinatorial optimization problem. In addition to this, classical heuristics build the solution incrementally, based on simple ideas, so problems such as non-convexity or non-smoothness do not affect their behavior significantly. The main idea, for these heuristics, is to select the next movement from a list of available movements usually according to a greedy criterion. For example, the Clarke and Wright heuristic for the VRP selects the edge with the highest savings, while the NEH heuristic for PFSP takes the job with the largest total processing time. Our proposal is to introduce a biased random behavior in the selection step, but still take into account the "common sense" rules enforced by the deterministic heuristics. More exactly, instead of having a single choice at every step, we will have multiple choices, each with a different probability of being chosen.

It is not hard to notice that our approach is similar to GRASP, but there are several important differences. We do not restrict the set of movements list of candidates with a parameter, and we do not select uniformly at random. Therefore, a better label for our methodology would be "Biased GRASP". As you can notice, the two probability distributions that we have used have been the geometric for the VRP and a discrete version of the triangular distribution for the FSP. We present these examples next, as they represent a direct application of our methodology.

TABLE 5. MSCH algorithm

```
Procedure: MSCH
begin
    procedure MSCH(maxIt, seed)
        ReadInputFromClassicalHeuristic();
        for i := 1 to maxIt do
            Solution:=GreedyNotUniformRandomize(seed);
            Solution:=LocalSearch(Solution);
            UpdateSolution(Solution);
        end for;
        return BestSolution;
    end MSCH
```

6. NUMERICAL RESULTS

Some specific examples of this technique are analyzed to illustrate the main ideas behind the method. In particular, we show how a geometric distribution can be combined with the Clarke and Wright (1964) heuristic to compute efficient solutions for the Vehicle Routing Problem, and how a discrete version of the triangular distribution can be incorporated into the NEH heuristic of Nawaz et alt. (1983) to attack the Permutation Flowshop Sequencing Problem.

6.1. **Applying MSCH to the Vehicle Routing Problem.** Our first example is that of the Vehicle Routing Problem (VRP), a well-known combinatorial optimization problem and probably the most popular one in the routing field. Due to its wide range of applications, VRP has stimulated the interest of a large number of researchers over the past decades. Even recently, a considerable amount of effort has been put into solving efficiently different versions of the problem. In fact, according to Eksioglu et al. (2009), the number of papers related to this topic has grown exponentially in the last 50 years. In VRP, the demands of a given set of customers must be satisfied using a homogeneous fleet of vehicles. All the routes must start and end at a depot node, where all the resources are initially located. The goal is to find an optimal set of routes, such that all the customers receive the supplies they need. An example of a set of routes for a particular VRP topology is given in Figure 3. There are many variants of the VRP, depending on the constraints that are enforced. In this section we focus on the Capacitated Vehicle Routing Problem (CVRP) which is described in detail in the comprehensive book dedicated to the VRP by Toth and Vigo (2002). For the sake of completeness, we mention here the constraints that are characteristic to the CVRP case:

1) all routes begin and end at a depot node,
2) each non-depot node must be supplied by exactly one vehicle,

3) a vehicle cannot stop twice at the same non-depot node and
4) No vehicle can be loaded exceeding its maximum capacity.

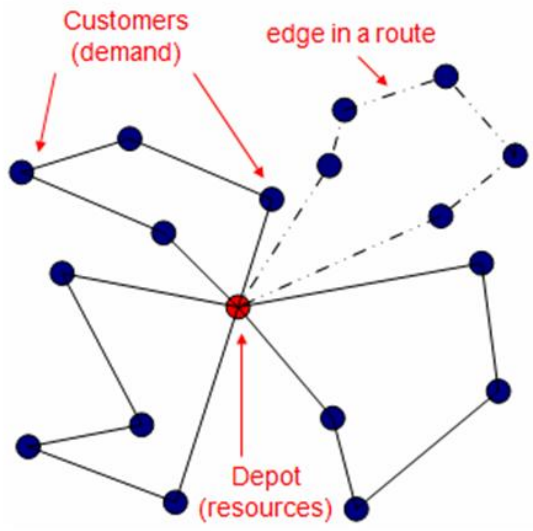

FIGURE 3. A sample of routes for a VRP instance

A variety of approaches have been tried for solving the CVRP. Berger and Barkaoui (2003) proposed a hybrid genetic algorithm in which two populations of solutions were concurrently evolving. Mazzeo and Loiseau (2004) developed an Ant Colony algorithm using techniques first introduced by Colorni, Dorigo and Maniezzo (1996). Lin et al (2007) [could not find this reference] employed simulated annealing methods and combined them with a local search to generate solutions for the CVRP. In addition to this, quite a few researchers (Osman, 1993; Taillard, 1993; Gendreau et al., 1994) have designed algorithms that rely on tabu search ideas.

Our hybrid algorithm, which we have called SR-GCWS (Simulation in Routing via the Generalized Clarke and Wright Savings heuristic), combines Monte Carlo simulation (MCS) with the Clarke and Wright heuristic (CWS) and takes advantage of the recent improvements in the field of random number generators (L'Ecuyer (2006)). More exactly, we use the parallel version of CWS, as described in [MIT website reference]. This heuristic is based on a very simple yet powerful idea, and selects at each step the edge with the highest savings. Our method generalizes this approach and assigns a nonzero probability of being chosen to each edge in the savings list. We do take into account the original greedy idea enforced by the CWS heuristic so we assign higher probabilities to the edges that give more savings. Therefore, we basically introduce a random biased behavior in the CWS heuristic by means of a probability distribution. Since simplicity was one of our main design goals, we aimed to generate the randomness necessary

for this process using as few parameters as possible. This is why we decided to employ quasi-geometric distributions that are modeled by a single parameter?. As a result, we manage to avoid a complex and time-consuming tuning process. A different quasi-geometric distribution is used every time we select an edge from the list. We have experimented with different values for the parameter α, and we have found out that the best result are obtained when $0.1 < \alpha < 0.2$. As you can see in the graph below, the smaller the ?, the slower the total cumulative distribution gets very close to 1, so more edges have a relatively high probability of being selected.

By following this simple method, solutions that outperform the CWS one are obtained in just a few iterations. As a result, hundreds of good solutions are obtained within seconds. However, even though the randomization process greatly reduces costs, a second push is needed in order to reach the best-known solution (BKS). To get there, we incorporate two improvements into our algorithm. First of all, we maintain a hash table that keeps track of the best route for any given set of nodes. This memory structure is helpful for improving the new solutions that are generated and it works very fast. Our second improvement is a splitting strategy that could be regarded as an example of a "divide-and-conquer" approach. Given a global solution, the instance is sub-divided in several smaller instances and then the algorithm is reapplied on each of them. This part is slow compared to the first enhancement, but it provides significant improvements.

We have implemented our methodology as a Java application and tested it on 50 classical CVRP benchmark instances selected from the web *www.branchandcut.org*. All the tests have been run on a personal computer (Intel CoreTM 2 Duo at 2.4 GHz and 2 GB RAM). It is also worth mentioning that our algorithm performed well in a variety of scenarios. For example, the efficiency of our method did not decrease as we changed from a common depot at the center topology to depot at one corner or to a cluster topology. Table 6 summarizes the results and makes a comparison between the solution generated by our algorithm, SR-GCWS, and the BKS listed on *www.branchandcut.org*. We obtain a negative average gap, which means that at least in some of the cases our algorithm improves the BKS. For some of the instances, though, the negative gap results because integer rounding has been used to compute the best-know solutions. We consider these results encouraging, especially given that they have been obtained in just a few seconds using an interpreted language such as Java. Finally, it is important to point out that our algorithm needs little instantiation and requires almost no fine-tuning processes. For a more detailed discussion of the methodology, the reader can consult (Juan et al, 2010).

6.2. **Applying MSCH to the Permutation Flowshop Sequencing Problem.** The second example we give to illustrate our methodology is that of the Permutation Flow-shop Sequencing Problem (PFSP), a well-known scheduling problem. The task now is to process a set J on n independent jobs on a set

TABLE 6. Comparison of methodologies for six randomly selected CVRP instances

Instance	Number of nodes	CWS solution	Best-known solution*	Our best solution	Gap
$A-n45-k7$	45	1,199.98	1,147.28	1,146.91	−0.03%
$A-n60-k9$	60	1,421.88	1,355.80	1,355.80	0.00%
$A-n80-k10$	80	1,860.94	1,766.50	1,766.50	0.00%
$B-n50-k7$	50	748.80	744.78	744.23	−0.07%
$B-n52-k7$	52	764.90	750.08	749.97	−0.01%
$B-n57-k9$	57	1,653.42	1,603.63	1,602.29	−0.08%
$B-n78-k10$	78	1,264.56	1,229.27	1,228.16	−0.09%
$E-n51-k5$	51	584.64	524.94	524.61	−0.06%
$E-n76-k10$	76	900.26	837.36	839.13	0.21%
$E-n76-k14**$	76	1,073.43	1,026.71	1,026.14	−0.06%
$F-n135-k7$	135	1,219.32	1,170.65	1,170.33	−0.03%
$M-n121-k7$	121	1,068.14	1,045.16	1,045.60	0.04%
$P-n70-k10$	70	896.86	830.02	831.81	0.22%
$P-n101-k4$	101	765.38	692.28	691.29	−0.14%
Average gap					−0.01%

M of m independent machines. Each job has a certain processing time on each of the machines. In general, we will say that job j requires pij units of time to be completed on machine i. There are also several restrictions that must be considered:

1) the execution of a job cannot be interrupted,
2) Each machine can execute at most one job at a time and
3) the order in which the jobs are processed on the machines is the same.

The goal is to find a permutation of the jobs that is optimal given a certain criterion. The most common criterion is the minimization of the total completion time, also referred to as the makespan or Cmax. The resulting problem is formally denoted as Fm—prmu—Cmax. The best solution is one (or more) of the n! possible permutation sequences, and it has been shown that the problem is NP-complete in the general case (Rinnooy Kan, 1976). Figure 7 depicts one possible scheduling of 3 jobs on 3 machines.

A lot of interest has been shown for the PFSP and many heuristics and metaheuristics have been developed. Johnson (1954) was the first to attack this problem. His contribution is extremely important since he has constructed a simple

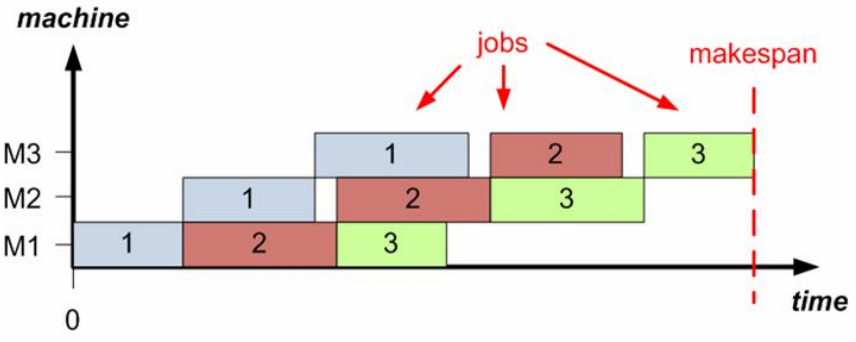

FIGURE 4. Example of a PFSP instance with 3 jobs and 3 machines

method to determine the optimal sequence of jobs when there are just two machines. This is one example of applying a restriction to a problem so that you are then able to solve it exactly in polynomial time. There have been numerous heuristics for solving the PFSP with more than two machines, but we will mention here just the one by Nawaz et al. (1983) known in the literature as the NEH heuristic. This one is commonly considered to be the best performing heuristic for PFSP. NEH is based on the common sense idea of trying to schedule first the most demanding jobs, i.e. the one with larger total processing time. It starts by creating an efficiency list of jobs sorted in decreasing order according to their total completion time. Then, at each step, the job at the top of the efficiency list is removed and put in a new list in the position that minimizes the partial makespan. The original running time of this procedure is O(n3m), where n is the number of jobs and m is the number of machines, but it can be reduced to O(n2m) by applying a technique known as Taillard acceleration (Taillard (1990)). In our approach we use the data structure introduced by Taillard since it is fairly straightforward to implement.

There are also several metaheuristics for the PFSP that explore a variety of ideas and techniques. Osman and Potts (1989) used simulated annealing. Widmer and Hertz (1989) proposed SPIRIT, a Tabu Search algorithm that made use of the NEH heuristic. Chen et al. (1995) employed genetic algorithms to solve the PFSP. Moreover, Chandrasekharan and Ziegler (2003) introduce an Ant Colonization Optimization algorithm for this problem. These are just a few examples that we mention to point out the wide range of methods that have been used to attack the PFSP. All these approaches are fairly easy to implement and can also be adapted to other variants of the flowshop problem. On the other hand, there are some hybrid algorithms that may find slightly better solutions, but are so complex and sophisticated that coding them is almost an impossible task or does not give the expected results. To avoid this problem, we have again tried to make our algorithm, SS-GNEH, as simple as possible.

Our method is similar to the GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristics introduced by Feo and Resende (1995) and diescribed in section 4.4. Unlike GRASP or other metaheuristics, our algorithm is parameter-free so it avoids complicated fine-tuning processes. The main idea is to combine Monte Carlo Simulation with the NEH heuristic and then start an iterative process to search the space of solutions. We choose a discrete triangular distribution like the one depicted in Figure 8 to introduce a random biased behavior in the algorithm.

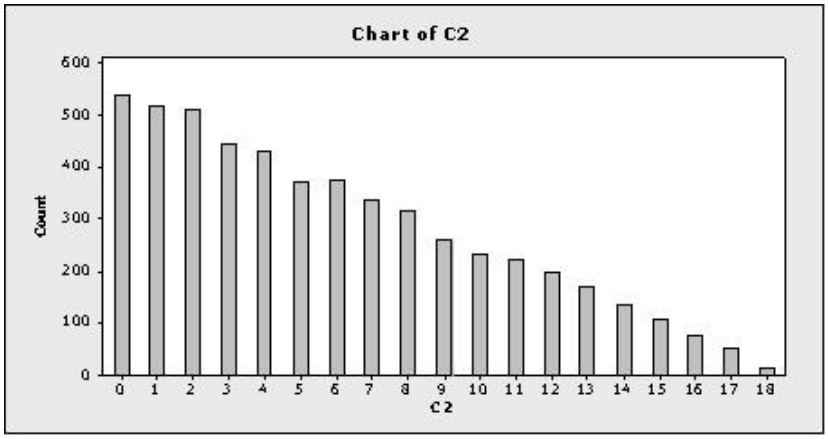

FIGURE 5. Assigning probabilities to 19 jobs using a discrete version of the triangular distribution

The goal is to improve the NEH solution by slightly perturbing it. To do so, we choose the jobs according to the probability distribution rather than their total completion time, as NEH does. The main steps of the algorithm could be summarized as follows:

1) generate Randomized NEH solutions until you find one (the base) that outperforms the original NEH solution,
2) apply a local search to the base solution found in step (1) as long as you get improvements,
3) update the best solution found so far if necessary,
4) restart the entire process if time permits (the threshold we use is 30 ms x number of jobs x number of machines) and
5) run each instance with several different seeds for the random number generator.

In the local search phase we pick at random and without repetition a job from the list n times, where n is the number of jobs, and then apply Taillard acceleration to find the best position for it. Note that this operation can only improve the base solution. However, if we would stop the algorithm at this point, we would likely end up in the vicinity of a local optimum that can still be far away from

the global optimum. To avoid this trap, we restart the entire process so we have to update the best solution found so far if necessary.

The methodology presented above has been implemented as a Java application. To run the code we have used an Intel Xeon at 2 GHz and 4 GB of RAM. The environment for running the code was the Eclipse IDE for Java under Windows 7. We tested the first 95 instances from Taillard (1993) that can be found at http://mistic.heig-vd.ch/taillard/default.htm, and we compared our results with the best-know solution (BKS), as reported by Zobolas et al. (2009). Table 2 gives a summary of the results, both for an earlier version of our algorithm (SS-GNEH-6b) and a more recent one in which a second local search phase has been added (SS-GNEH-6c). Since we run each instance 15 times, with different seeds, we compute both an average gap with respect to the BKS, as well as a minimum gap.

TABLE 7. Summary of Results for Taillard's instances

	Average Gap				Minimum Gap			
	$6b$	$6c$	$IGd4T04$	$IGd8T02$	$6b$	$6c$	$IGd4$	$IGd8T02$
$Tai1-30$	0.07%	0.05%	0.05%	0.04%	0.00%	0.01%	0.01%	0.00%
$Tai31-60$	1.15%	0.74%	0.57%	0.60%	0.92%	0.52%	0.34%	0.37%
$Tai61-90$	0.91%	0.62%	0.53%	0.59%	0.75%	0.44%	0.29%	0.39%
$Tai90-95$	0.26%	0.18%	0.21%	0.20%	0.19%	0.09%	0.07%	0.08%

We also compare our solutions with the ones produced by a state-of-the-art algorithm, namely the Iterated Greedy algorithm or shortly the IG (Ruiz and Stutzle, 2007). In the graphs from Figure 10, the two upper lines represent versions of our method, while the other two correspond to the IG. It can be easily noticed by analyzing the 2 graphs above that the better version of our algorithm (6c) comes very close to the IG, a fact that we consider encouraging given the simplicity of our methodology. As we have previously mentioned, our method needs little instantiation and does not require any complicated fine-tuning processes. The reader is referred to Juan et al. (2010) for a more detailed explanation of our approach.

7. CONCLUSIONS

In this paper, stochastic algorithms are proposed as a method for solving non-smooth combinatorial optimization problems. The key idea in our approach is to employ probability distributions such as the geometric or the triangular, to add a random biased behavior to classical heuristics. This way we obtain a large set of alternative good solutions that outperform the initial solution produced by the heuristic. An overview of non-convex and non-smooth optimization problems has

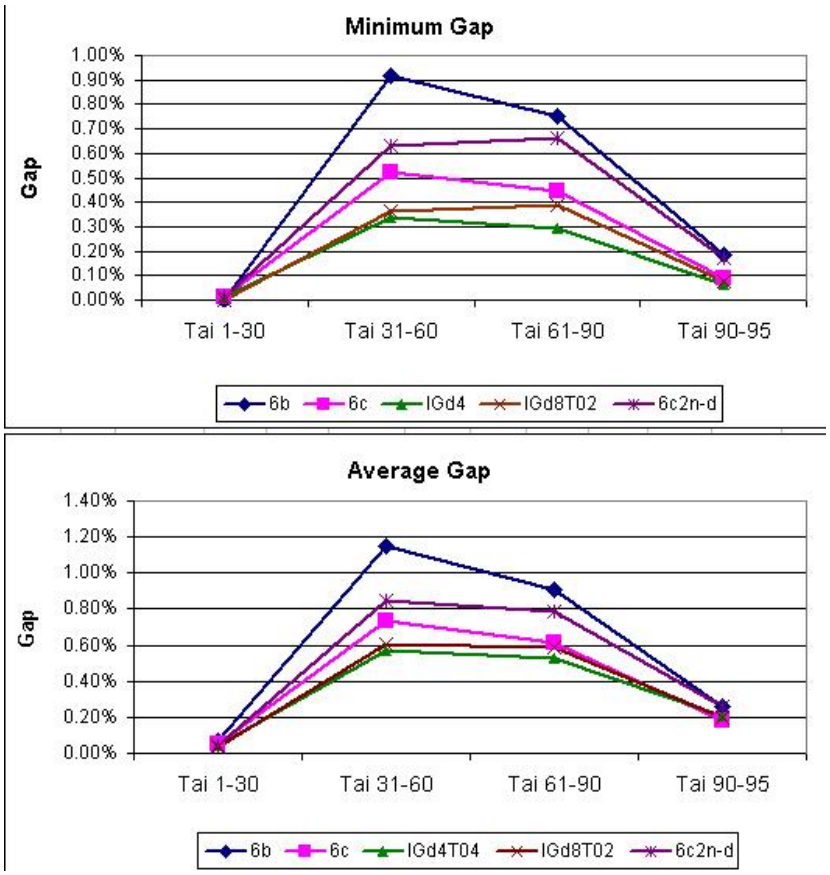

FIGURE 6. Graphs comparing SS-GNEH with the IG for the first 95 Taillard instances

been given to introduce the reader to the topic. We have also discussed about the traditional methods for solving this type of problems, such as genetic and evolutionary algorithms, tabu search, or simulated annealing, with the purpose of giving an overall picture. As it has been pointed out, our methodology has similarities with the GRASP metaheuristic, but there are important differences that have already been discussed. For this reason, a better label for our methodology would be "Biased GRASP". We have employed the idea of randomizing classical heuristics to find efficient solutions for two well-known combinatorial optimization problems, the Capacitated Vehicle Routing Problem and the Permutation Flowshop Sequencing Problem. By applying our methodology we have obtained competitive results, and even reached the best know-solution in many of the instances that we have tested. We consider these promising results a proof of our method's potential, and we believe that the same ideas can be useful in a variety of other settings.

References

[1] Ali, M.K., and F. Kamoun, 1993. *Neural Network for Shortest Path Computation and Routing in Computer Networks*. IEEE Transaction Neural Networks, 4, 941–953.

[2] Al-Sultan, K.,S., 1995. *A tabu search approach to the clustering problem*. Pattern Recognition, 28 (9), 1443–451.

[3] Bagirov, A., M. and J. Yearwood, 2006. *A new nonsmooth optimization algorithm for minimum sum-of-squares clustering problems*. Eur. J. Oper. Res., 170, 578–596.

[4] Berger, J. and M. Barkaou, 2003. *A hybrid genetic algorithm for the capacitated vehicle routing problem*. In: Proceedings of the International Genetic and Evolutionary Computation Conference - GECCO03, LNCS 2723, 646–656.

[5] Boyd S. and L. Vandenberghe, 2004. *Convex Optimization*. Cambridge University Press, Cambridge.

[6] Chandrasekharan, R., and H. Ziegler, 2004. *Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs*. European Journal of Operational Research 155(2), 426–438.

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

[8] Clarke G. and J. Wright, 1964. *Scheduling of vehicles from a central depot to a number of delivery points*. Operations Research 12 (4), 568–581.

[9] Dorigo, M., Maniezzo, V. and A. Colorni, 1996. *Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26 (1), 29–41.

[10] Eksioglu, B., Vural, A.,V. and A.Reisman, 2009. *The vehicle routing problem: A taxonomic review. Computers and Industrial Engineering*, `doi:10.1016/j.cie.2009.05.009`.

[11] Feo, T.,A. and M.G.C. Resende, 1989. *A probabilistic heuristic for a computationally difficult set covering problem*. Operations Research Letters, 8, 67–71.

[12] Gendreau, M., Hertz, A. and G. Laporte, 1994. *A Tabu Search Heuristic for the Vehicle Routing Problem*. Management Science, 40, 1276–1290.

[13] Glover, F. and M. Laguna, 1997. *Tabu Search*. Kluwer Academic Publishers, Boston.

[14] Glover, F., 1986. *Future paths for integer programming and links to artificial intelligence*. Computers and Operations Research, 13,533–549.

[15] Hamdan, M. and M.E. El-Hawary, 2002. *Hopfield-Genetic Approach for Solving the Routing Problem In computer Networks*. Proceedings of the 2002 IEEE Canadian Conference on Electrical and Computer Engineering, 823–827.

[16] Holland, J.H., 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor Johnson, S.M., 1954. *Optimal two- and three-stage production schedules with setup times included*. Naval Research Logistics Quarterly 1, 61–68.

[17] Juan, A., Faulin, J., Ruiz, R., Barrios, B. and S. Caballe. *The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem*. Applied Soft Computing. Juan, A., Ionescu, D., Ruiz, R., Mateo, M. and H. Loureno, 2010.

[18] *A Simulation-based Approach for Solving the Flowshop Problem*. In: Proceedings of the 2010 Winter Simulation Conference. Baltimore, Maryland, USA. December 5–8.

[19] Kirkpatrick, S., Jr., C. G., and M. Vecchi, 1983. *Optimization by simulated annealing*. Science, 220. L.J. Fogel, 1962. Autonomous automata. Industrial Research, 4, 14–19.

[20] L'Ecuyer, P., 2006. *Random Number Generation in Simulation*. Elsevier Science, Amsterdam.

[21] Mazzeo, S. and I. Loiseau, 2004. *An Ant Colony Algorithm for the capacitated vehicle routing problem*. Electronic Notes in Discrete Mathematics, Elsevier, 18, 181–186.

[22] Metropolis,N., Rosenbluth, A., Rosenbluth, M., Teller, A., and E. Teller, 1953. *Equation of State Calculations by Fast Computing Machines*. J. Chem. Phys.,21 (6), 1087–1092.
[23] Nawaz, M., Enscore, Jr, E. E., and I. Ham, 1983. *A heuristic algorithm for the m-machine, n-job flowshop sequencing problem*. OMEGA, The International Journal of Management Science, 11(1), 91–95.
[24] Oonsivilai, A., Srisuruk, W., Marungsri, B. and T. Kulworawanichpong, 2009. *Tabu Search Approach to Solve Routing Issues in Communication Networks*. World Academy of Science, Engineering and Technology, 1174–1177.
[25] Osman, I.,H., 1993. *Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem*. Annals of Operations Research, 41, 421–451.
[26] Osman, L., and C. Potts, 1989. *Simulated annealing for permutation flow-shop scheduling*. OMEGA, 17 (6), 551–557.
[27] Rechenberg, I., 1973. *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany.
[28] Rinnooy Kan, A.H.G., 1976. *Machine Scheduling Problems: Classification, Complexity and Computations*. Springer.
[29] Rojas, R., 1996. *Neural Networks: A Systematic Introduction*. Springer-Verlag, Heidelberg, Germany.
[30] Ruiz, R., and T. Sttzle, 2007. *A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem*. European Journal of Operational Research 177, 2033–2049.
[31] Selim, S., Z. and K.S. Al-Sultan, 1991. *A simulated annealing algoroithm for the clustering problem*. Pattern Recognition, 24 (10), 1003–1008.
[32] Taillard, E., 1990. *Some efficient heuristic methods for the flow shop sequencing problem*. European Journal of Operational Research 47, 65–74.
[33] Taillard, E., 1993. *Benchmarks for basic scheduling problems*. European Journal of Operations Research, 64, 278–285.
[34] Taillard, E., D., 1993. *Parallel Iterative Serach Methods for Vehicle Routing Problem*. Networks, 23, 661–673.
[35] V. Ramadurai and M. L. Sichitiu, 2003. *Localization in wireless sensor networks: a probabilistic approach*. International conference on Wireless Networks (ICWN03), 275–281.
[36] Widmer, M., and A. Hertz, 1989. *A new heuristic method for the flow shop sequencing problem*. European Journal of Operational Research, 41(2), 186–193.
[37] Wright, S., 1997. *Primal-Dual Interior-Point Methods*. SIAM.
[38] Zobolas, C., Tarantilis, C., and G. Ioannou, 2009. *Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm*. Computers and Operations Research 36, 1249–1267.

Dragos Ionescu
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139 – 4307
USA

Angel A. Juan
Computer Science Department
Open University of Catalonia - IN3
08018 Barcelona
Spain

Javier Faulin
Statistics and Operations Research Department
Public University of Navarre
31006 Pamplona
Spain

Albert Ferrer
Applied Mathematics 1 Department
Technical University of Catalonia
08028 Barcelona
Spain