Biased Randomization of Classical Heuristics

Angel A. Juan
IN3-Open University of Catalonia, Spain

José Cáceres-Cruz
IN3-Open University of Catalonia, Spain

Sergio González-Martín
IN3-Open University of Catalonia, Spain

Daniel Riera
IN3-Open University of Catalonia, Spain

Barry B. Barrios
IN3-Open University of Catalonia, Spain

INTRODUCTION

In the context of combinatorial optimization problems, this chapter discusses how to randomize classical heuristics in order to transform these deterministic procedures into more efficient probabilistic algorithms. This randomization process can be performed by using a uniform probability distribution or, even more interesting, by using a non-symmetric distribution.

Combinatorial Optimization Problems (COPs) have posed numerous challenges to the human mind throughout the past decades. 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 that must be satisfied. From a theoretical point of view, these problems have an interest on their own due to the mathematics involved in their modeling, analysis and solution. However, the main reason for which they have been so actively investigated is the tremendous amount of real-life applications that can be successfully modeled as a COP. Thus, for example, decision-making processes in fields such as logistics, transportation, and manufacturing contain plentiful hard challenges that can be expressed as COPs (Faulin et al., 2012; Montoya et al., 2011). Accordingly, researchers from different areas –e.g. Applied Mathematics, Operations Research, Computer Science, and Artificial Intelligence– have directed their efforts to conceive techniques to model, analyze, and solve COPs.

A considerable number of methods and algorithms for searching optimal or near-optimal solutions inside the solution space have been developed. In some small-sized problems, the solution space can be exhaustively explored. For those instances, efficient exact methods can usually provide the optimal solution in a reasonable time. Unfortunately, the solution space in most COPs is exponentially astronomical. Thus, in medium- or large-size problems, the solution space is too large and finding the optimum in a reasonable amount of time is not a feasible task. An exhaustive method that checks every single candidate in the solution space would be of very little help in these cases, since it would take exponential time. Therefore, a large amount of heuristics and metaheuristics have been developed in order to obtain near-optimal solutions, in reasonable computing times, for medium- and large-size problems, some of them even considering realistic constraints.

DOI: 10.4018/978-1-4666-5202-6.ch028

The main goal of this chapter is to present a hybrid scheme which combines classical heuristics with biased-randomization processes. As it will be discussed later, this hybrid scheme represents an efficient, relatively simple, and flexible way to deal with several COPs in different fields, even when considering realistic constraints.

BACKGROUND

In the context of this chapter, we will refer to any algorithm which makes use of pseudo-random numbers to perform ‘random’ choices during the exploration of the solution space by the term randomized search method, or simply randomized algorithm. This includes most current metaheuristics and stochastic local-search processes. Thus, since it does not follow a determinist path, even for the same input, a randomized search method can produce different outputs in different runs. Within these type of algorithms we can include, among others, the Genetic and Evolutionary Algorithms (Reeves, 2010), Simulated Annealing (Nikolaev & Jacobson, 2010), Greedy Randomized Adaptive Search Procedure or GRASP (Festa & Resende, 2009a, 2009b), Variable Neighborhood Search (Hansen et al, 2010), Iterated Local Search (Lourenço et al., 2010), Ant Colony Optimization (Dorigo &Stützle, 2010), Probabilistic Tabu Search (Lokketangen & Glover, 1998), or Particle Swarm Optimization (Kennedy & Eberhart, 1995).

One of the most popular randomized search methods is GRASP (Resende & Ribeiro 2010). Roughly speaking, GRASP is a multi-start or iterative process which uses uniform random numbers and a restricted candidate list to explore the solution space (Figure 1). At each iteration, two phases are executed: (a) the construction phase, which generates a new solution by randomizing a classical heuristic; and (b) a local search phase, which aims at improving the previously constructed solution. At the end of this multi-start process, the best found solution is kept as the result.

It is interesting to notice that most of the work on randomized algorithms is based on the use of ‘uniform randomness’, i.e.: randomness is generated throughout a symmetric (non-biased) uniform distribution. Thus, when we talk about biased randomization, we refer to the use of probability distributions –other than the uniform– which do not distribute probabilities in a symmetric shape but in a skewed one. Of course, these non-symmetric (skewed) distributions can also be used to induce ‘biased randomness’ into an algorithm. As a matter of fact, as far as we know, the first approach based on the use of biased randomization of a classical heuristic is due to Bresina (1996). This author proposes a methodology called Heuristic-Biased Stochastic Sampling (HBSS), which performs a biased iterative sampling of the search tree according to some heuristic criteria. Bresina applies the HBSS to a scheduling problem, and concludes that this approach outperforms greedy search within a small number of samples.

More recently, Juan et al. (2011a) proposed the use of non-symmetric probability distributions to induce randomness in classical heuristics. Their general framework was called Multi-start biased Randomization of classical Heuristics with Adaptive local search (MIRHA). On this approach, the authors propose to combine classical greedy heu-

Figure 1. General pseudo-code for GRASP

```
procedure GRASP(inputs)
  while stopping criterion is not satisfied do
     solution ← ConstructGreedyRandomizedSolution(inputs)
     solution ← ApplyLocalSearch(solution)
     bestSolution ← UpdateBestSolution(solution)
  endwhile
  return bestSolution
endprocedure
```

Related Content