Computers & Operations Research 68 (2016) 75–88

Contents lists available at ScienceDirect

Computers & Operations Research

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

Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization

Christian Blum [a,b,*], Pedro Pinacho [a,c], Manuel López-Ibáñez [d], José A. Lozano [a]

[a] Department of Computer Science and Artificial Intelligence, University of the Basque Country UPV/EHU, San Sebastian, Spain
[b] IKERBASQUE, Basque Foundation for Science, Bilbao, Spain
[c] Escuela de Informática, Universidad Santo Tomás, Concepción, Chile
[d] Alliance Manchester Business School, University of Manchester, UK

ARTICLE INFO

Available online 2 November 2015

Keywords:
Metaheuristics
Exact solver
Hybrid algorithms
Minimum common string partition
Minimum covering arborescence

ABSTRACT

This paper describes a general hybrid metaheuristic for combinatorial optimization labelled Construct, Merge, Solve & Adapt. The proposed algorithm is a specific instantiation of a framework known from the literature as Generate-And-Solve, which is based on the following general idea. First, generate a reduced sub-instance of the original problem instance, in a way such that a solution to the sub-instance is also a solution to the original problem instance. Second, apply an exact solver to the reduced sub-instance in order to obtain a (possibly) high quality solution to the original problem instance. And third, make use of the results of the exact solver as feedback for the next algorithm iteration. The minimum common string partition problem and the minimum covering arborescence problem are chosen as test cases in order to demonstrate the application of the proposed algorithm. The obtained results show that the algorithm is competitive with the exact solver for small to medium size problem instances, while it significantly outperforms the exact solver for larger problem instances.

1. Introduction

In this paper we introduce a general algorithm for combinatorial optimization labelled Construct, Merge, Solve & Adapt (CMSA). The proposed algorithm belongs to the class of hybrid metaheuristics [1–4], which are algorithms that combine components of different techniques for optimization. Examples are combinations of metaheuristics with dynamic programming, constraint programming, and branch and bound. In particular, the proposed algorithm is based on the following general idea. Imagine it were possible to identify a substantially reduced sub-instance of a given problem instance such that the sub-instance contains high-quality solutions to the original problem instance. This would allow applying an exact technique—such as, for example, a mathematical programming solver—with little computational effort to the reduced sub-instance in order to obtain a high-quality solution to the original problem instance. This is for the following reason. For many combinatorial optimization problems the field of mathematical programming—and integer linear programming (ILP) in particular—provides powerful tools; for a comprehensive introduction into this area see, for example, [5]. ILP-solvers are in general based on a tree search framework but further include the solution of linear programming relaxations of a given ILP model for the problem at hand (besides primal heuristics) in order to obtain lower and upper bounds. To tighten these bounds, various kinds of additional inequalities are typically dynamically identified and added as cutting planes to the ILP-model, yielding a branch & cut algorithm. Frequently, such ILP approaches are highly effective for small to medium sized instances of hard problems, even though they often do not scale well enough to large instances relevant in practice. Therefore, in those cases in which a problem instance can be sufficiently reduced, a mathematical programming solver might be very efficient in solving the reduced problem instance.

1.1. Related work

The general idea described above is present in several works from the literature. For example, it is the underlying idea of the general algorithm framework known as Generate-And-Solve (GS) [6–9]. In fact, our algorithm can be seen as an instantiation of this framework. The GS framework decomposes the original optimization problem into two conceptually different levels. One of the two levels makes use of a component called *Solver of Reduced Instances* (SRI), in which an exact method is applied to sub-instances of the original problem instance that maintain the conceptual structure of the original instance, that is,

* Corresponding author.
E-mail addresses: christian.blum@ehu.es (C. Blum), ppinacho@santotomas.cl (P. Pinacho), manuel.lopez-ibanez@manchester.ac.uk (M. López-Ibáñez), ja.lozano@ehu.es (J.A. Lozano).

http://dx.doi.org/10.1016/j.cor.2015.10.014
0305-0548/

any solution to the sub-instance is also a solution to the original instance. At the other level, a metaheuristic component deals with the problem of generating sub-instances that contain high quality solutions. In GS, the metaheuristic component is called *Generator of Reduced Instances* (GRI). Feedback is provided from the SRI component to the GRI component, for example, by means of the objective function value of the best solution found in a sub-instance. This feedback serves for guiding the search process of the GRI component.

Even though most existing applications of the GS framework are in the context of cutting, packing and loading problems—see, for example, [7–11]—other successful applications include the ones to configuration problems arising in wireless networks [12–14]. Moreover, it is interesting to note that the applications of GS published to date generate sub-instances in the GRI component using either evolutionary algorithms [10,14] or simulated annealing [11,13]. Finally, note that in [10] the authors introduced a so-called density control operator in order to control the size of the generated sub-instances. This mechanism can be seen as an additional way of providing feedback from the SRI component to the GRI component.

Apart from the GS framework, the idea of solving reduced problem instances to optimality has also been explored in earlier works. In [15,16], for example, the authors tackle the classical traveling salesman problem (TSP) by means of a two-phase approach. The first phase consists in generating a bunch of high-quality TSP solutions using a metaheuristic. These solutions are then merged, resulting in a reduced problem instance, which is then solved to optimality by means of an exact solver. In [17] the authors present the following approach for the prize-collecting Steiner tree problem. First, the given problem instance is reduced in such a way that it still contains the optimal solution to the original problem instance. Then, a memetic algorithm is applied to this reduced problem instance. Finally, a mathematical programming solver is applied to find the best solution to the problem instance obtained by merging all solutions of the first and the last population of the memetic algorithm. Massen et al. [18,19] use an ant colony optimization algorithm to generate a large number of feasible routes for a vehicle routing problem with feasibility constraints, then apply an exact solver to a relaxed set-partitioning problem in order to select a subset of the routes. This subset is used to bias the generation of new routes in the next iteration.

Finally, note that a first, specific, application of the general algorithm proposed in this work has been published in [20] in the context of the minimum weight arborescence problem.

1.2. Contribution of this work

Even though—as outlined above—there is important related work in the literature, the idea of iteratively solving reduced problem instances to optimality has not yet been explored in an exhaustive manner. In this work we introduce a generally applicable algorithm labelled Construct, Merge, Solve & Adapt (CMSA) for tackling combinatorial optimization problems. The algorithm can be seen as a specific instantiation of the GS framework. It is designed to take profit from ILP solvers such as CPLEX even in the context of large problem instances to which these solvers can not be applied directly. In particular, the main feature of the algorithm is the generation of sub-instances of the original problem instance by repeated probabilistic solution constructions, and the application of an ILP solver to the generated sub-instances. Hereby, the way of generating sub-instances by merging the solution components found in probabilistically constructed solutions distinguishes our algorithm from other instantiations of the GS framework from the literature. This feature is actually quite appealing, because our algorithm can easily be applied to any problem for which (1) a constructive heuristic and (2) an exact solver are known.

We consider two test cases for the proposed algorithm: (1) the *minimum common string partition* (MCSP) problem [21], and a *minimum covering arborescence* (MCA) problem, which is an extension of the problem tackled in [20]. For both problems, ILP solvers such as CPLEX are very effective in solving small to medium size problem instances. However, their performance deteriorates (1) in the context of the MCSP problem when the length of the input strings exceeds 600, and (2) in the context of the MCA problem when the number of nodes of the input graph exceeds 1000. We will show that the CMSA algorithm is a new state-of-the-art algorithm for the MCSP problem, especially for benchmark instances for which the application of CPLEX to the original ILP model is not feasible. In the context of the MCA problem we will show that our algorithm is able to match the performance of CPLEX for small and medium size problem instances. Moreover, when large size instances are tackled, the algorithm significantly outperforms a greedy approach.

1.3. Organization of the paper

The remaining part of the paper is organized as follows. The CMSA algorithm is outlined in general terms in Section 2. The application of this algorithm to the minimum common string partition problem is described in Section 3, whereas its application to the minimum covering arborescence problem is outlined in Section 4. An extensive experimental evaluation is provided in Section 5. Finally, in Section 6 we provide conclusions and an outlook to future work.

2. Construct, Merge, Solve & Adapt

In the following we assume that, given a problem instance $\mathcal{I}$ to a generic problem $\mathcal{P}$, set C represents the set of all possible components of which solutions to the problem instance are composed. C is henceforth called the complete set of solution components with respect to $\mathcal{I}$. Note that, given an integer linear (or non-linear) programming model for problem $\mathcal{P}$, a generic way of defining the set of solution components is to say that each combination of a variable with one of its values is a solution component. Moreover, in the context of this work a valid solution S to $\mathcal{I}$ is represented as a subset of the solution components C, that is, $S \subseteq C$. Finally, set $C' \subseteq C$ contains the solution components that belong to a restricted problem instance, that is, a sub-instance of $\mathcal{I}$. For simplicity reasons, C' will henceforth be called a sub-instance. Imagine, for example, the input graph in case of the TSP. The set of all edges can be regarded as the set of all possible solution components C. Moreover, the edges belonging to a tour S—that is, a valid solution—form the set of solution components that are contained in S.

The Construct, Merge, Solve and Adapt (CMSA) algorithm works roughly as follows. At each iteration, the algorithm deals with the incumbent sub-instance C'. Initially this sub-instance is empty. The first step of each iteration consists in *generating* a number of feasible solutions to the original problem instance $\mathcal{I}$ in a probabilistic way. In the second step, the solution components involved in these solutions are added to C' and an exact solver is applied in order to *solve* C' to optimality. The third step consists in *adapting* sub-instance C' by removing some of the solution components guided by an aging mechanism. In other words, the CMSA algorithm is applicable to any problem for which (1) a way of (probabilistically) generating solutions can be found and (2) a strategy for solving the problem to optimality is known.

In the following we describe the CMSA algorithm, which is pseudo-coded in Algorithm 1, in more detail. The main loop of the proposed algorithm is executed while the CPU time limit is not reached. It consists of the following actions. First, the best-so-far solution S_{bsf} is initialized to NULL, and the restricted problem instance (C') to the empty set. Then, at each iteration a number of n_a solutions is probabilistically generated (see function ProbabilisticSolutionGeneration(C) in line 6 of