A memetic algorithm for the minimum weighted k-cardinality tree subgraph problem*

Maria J. Blesa * Pablo Moscato † Fatos Xhafa *

* Departament de LSI, Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6, Jordi Girona, 1-3, 08034 Barcelona, Spain
Email: {mjblesa, fatos}@lsi.upc.es

† Faculdade de Engenharia Eletrónica e de Computação, Universidade Estadual de Campinas
C.P. 6101, Campinas, SP, CEP 13083-970, Brazil
Email: moscato@cacr.caltech.edu, moscato@bestway.com.br

1 Introduction and Motivation

The minimum weighted k-cardinality tree subgraph problem consists in finding, in a given undirected weighted graph $G(V, E, W)$, a tree $T \subseteq E$ of k edges having minimal total weight among all of k-trees that are subgraphs of G. We will refer this problem as the MIN W-k-TREE. This is combinatorial optimization problem first described by Hamacher, Jörnsten, and Maffioli in [HJM91] who also proved to be strongly NP-hard. This said, it is unlikely to be solved efficiently. Given this observation, researchers have focused on heuristic and metaheuristic algorithms to find suboptimal feasible solutions for the problem, as a good way to cope with most practical setting applications. The problem is of relevance to both theory and industrial applications. In the latter group we can cite *field leasing* [HJ93] were it has arisen. Other applications of this problem can be found in diverse areas such as *facility layout* and *graph partitioning* [FH92, FHW98], *open pit mining* [FHW97], *quorum-cast routing* [TCK94], *telecommunications* [GH97] and *matrix decomposition* [BFM98, BFM97]. We refer the reader to [JL97] for a general complete mathematical formulation of the problem as well as [EFHM97].

Among the first metaheuristics to be applied to it, we can mention the Tabu Search (TS) developed by Jörnsten and Løkketangen in [JL97]. Tabu Search method for the k-Cardinality Tree. Blesa and Xhafa have proposed another implementation of the TS method in [BX00a, BX00b] and compared their results with the ones in [EFHM97, JL97].

To our knowledge, no memetic algorithm (MA) has yet been reported for this problem. It is known that some MAs have a good synergy with TS when they use it as individual steps for diversification and local optimization by the agents. As a consequence, one of our main motivations was, first, to obtain a new implementation of an MA with as *little effort* as possible, using the ideas of *generic programming*, and that it should also be *flexible*, meaning that the basic MA would be easy to adapt without changing the implementation as a whole. A set of benchmarking tests of an MA on the same set of instances tested with the TS would help us to design a *generic skeleton*, being, hopefully as good as the TS implementation. We envision as a later step to adapt these two generic skeletons to implement an MA with periods of TS by each agent in the population. This will help addressing the issues of *flexibility*, *robustness*, *scalability*, as well as a considerable savings in time in development of

*Submitted to MIC'2001 - 4th Metaheuristics International Conference. Porto, Portugal, July 16-20, 2001

code, which are pretty much a concern when addressing larger instances.

2 Memetic Algorithm template

A generic skeleton for Local-Search based MAs was proposed by Moscato in [M99]. The pseudocode given below is generic enough to encompass most implementations which can be found in the literature.

```
Local-Search-based Memetic Algorithm

begin
    initializePopulation Pop using FirstPop();
    foreach i ∈ Pop do i := Local-Search-Engine(i);
    foreach i ∈ Pop do Evaluate(i);
    repeat /* generations loop */
        parfor j := 1 to #recombinations do
            selectToMerge a set S_par ⊆ Pop;
            offspring := Recombine(S_par, x);
            offspring := Local-Search-Engine(offspring);
            Evaluate(offspring);
            addInPopulation offspring to Pop;
        endparfor;
        parfor j := 1 to #mutations do
            selectToMutate i ∈ Pop;
            i_m := Mutate(i);
            i_m := Local-Search-Engine(i_m);
            Evaluate(i_m);
            addInPopulation i_m to Pop;
        endparfor;
        Pop := SelectPop(Pop);
        if Pop meetsPopConvCriteria then Pop := RestartPop(Pop);
    until termination-condition=True;
end;
```

Figure 1: The Local-Search-Based Memetic Algorithm

By means of this generic template, the proposed algorithm for the MIN W-k-TREE just needs to get defined some extra information. We need to define the problem-dependent methods FirstPop(), Local-Search-Engine(), Evaluate(), Recombine(), Mutate(), RestartPop(). We will also need to instantiate **initializePopulation**, **selectToMerge**, **meetsPopConvCriteria** (and the termination-condition for the algorithm), **addInPopulation**. These methods are in boldface type to indicate that they are not problem dependent. However, they instantiate different types of MAs as we will remark later. We will also identify the value of the numerical parameters. There are several memetic algorithms that use a hierarchically structured population. In this case, since we are interested in having a simpler code for a preliminary set of tests, we will use an unstructured population.

2.1 Problem-dependent methods

2.1.1 Evaluate() and FirstPop()

We will use as evaluation function (or guiding function) the same objective function, i.e. the sum of the weights of the k-edges of the tree. We create an initial population of 26 feasible solutions (k-cardinality trees) by means of a particular constructive heuristic, modified from DPT (DynamicPrimTree). The number of 26 has been chosen since we plan to later compare our results with an structured MA that uses a complete ternary tree with 13 agents, each one having two feasible solutions as in the paper by Buriol, França, and Moscato, who applied the method to the Asymmetric Traveling Salesman Problem [BFM99]. The first population is then constructed according to the pseudocode of Fig. 2

FirstPop()

begin
  repeat
  Select uniformly at random a vertex $v_0 \in V$;
  Apply Prim's algorithm starting at v_0 to find a minimum spanning tree $ST(v_0)$;
  Apply DynamicTree to $ST(v_0)$ to find a k-cardinality tree $T(v_0)$;
  Identify $L(T(v_0)) \subset V$, the set of vertices of V that are leaves of $T(v_0)$;
  foreach $v' \in L(T(v_0))$ **do**
    Apply Prim's algorithm starting at v' to find a minimum spanning tree $ST(v')$;
    Apply DynamicTree to $ST(v')$ to find a k-cardinality tree $T(v')$;
  endfor
  until 26 or more different k-cardinality trees have been generated
  Return the best 26 k-cardinality trees found.
end;

Figure 2: FirstPop() based on DynamicPrimTree

2.1.2 Local-Search-Engine(), Recombine(), and Mutate

For the individual optimization step of each agent we first selected the TSKTree (Tabu Search for Min W-k-Tree) instantiation of our framework as given in [BX00b]. In this case, the Local-Search-Engine involves only $|V|$ moves of the given feasible solution. However, agents do remember previous actions, so the two TabuLists (one of dropped and the other of added edges) are maintained between two consecutive individual optimization periods. There is one difference though, when new edges are added or deleted after recombination, mutation, or any other operation, we make the necessary changes in the TabuLists to indicate these updates.

The recombination operator has as input two feasible solutions and returns another feasible solution. It also uses a VertexTabuList structure for each agent to avoid repeating recombining from the same vertex. Let T_1 and T_2 be the two parent solutions, and let T_{off} be the "offspring" solution thus created. We first used the following recombination operator (Fig 3)

We also consider a simpler version (see Fig. 4), and hence two MAs, one with each type of recombination method are obtained.

The idea behind the mutation method is to include edges that can be out of the current population. We resort to the previous algorithms to do so (see Fig. 5). Since two types of recombination procedures have been considered, two types of mutation arise too. This results in a total of four combinations.

2.1.3 RestartPop()

This method is used when the population has lost diversity, and it is aimed to restart the search. We take the best k-tree found so far (the best one in the population) and we add it unaltered to the new population. We now need to create 25 new k-trees. We have denoted by $NRL \subset V$ the set of nodes that are neither roots nor leaves in the population of 26 trees that is passed as input.

Recombine() (first version)

begin
Let $G'(V', E') \subseteq G$ be the graph formed by the union of T_1 and T_2.
Select uniformly at random a vertex $v_0 \in V'$ that is not in the TabuList;
Apply Prim's algorithm starting at v_0 to find a minimum spanning tree $ST(v_0)$ of G';
Apply DynamicTree to $ST(v_0)$ to find a k-cardinality tree $T(v_0)$ of G';
Identify $L(T(v_0)) \subset V'$, the set of vertices of V' that are leaves of $T(v_0)$;
foreach $v' \in L(T(v_0))$ **do** (optional)
Apply Prim's algorithm starting at v' to find a minimum spanning tree $ST(v')$;
Apply DynamicTree to $ST(v')$ to find a k-cardinality tree $T(v')$;
endfor
Return as T_{off} best k-cardinality tree found.
Add vertex v_0 to the VertexTabuList of agents that have trees T_1 and T_2
end;

Figure 3: Recombine() based on DynamicPrimTree

Recombine() (simpler version)

begin
Let $G'(V', E') \subseteq G$ be the graph formed by the union of T_1 and T_2.
Select uniformly at random a vertex $v_0 \in V'$ that is not in the TabuList;
Apply Prim's algorithm starting at v_0 to find a minimum spanning tree $ST(v_0)$ of G';
Apply DynamicTree to $ST(v_0)$ to find a k-cardinality tree $T(v_0)$ of G';
Return as T_{off} the tree $T(v_0)$.
Add vertex v_0 to the VertexTabuList of agents that have trees T_1 and T_2
end;

Figure 4: A simpler Recombine() algorithm based on DynamicPrimTree

Mutate()

begin
Let T be the tree received as input;
Select uniformly at random a vertex $v_0 \in V(T)$ that is neither the root nor a leave;
Apply Prim's algorithm starting at v_0 to find a minimum spanning tree $ST(v_0)$ of G;
Apply DynamicTree to $ST(v_0)$ to find a k-cardinality tree $T(v_0)$ of G;
Return as T_{mut} the tree resulting from Recombine($T, T(v_0)$);
end;

Figure 5: Mutate() algorithm based on the simpler Recombine() method

2.2 Problem-independent Methods

2.2.1 Methods initializePopulation and addInPopulation

The **initializePopulation** just calls the FirstPop() methods. In the unstructured memetic algorithm, one feasible solution is assigned to each agent. The **addInPopulation** is in charge of verifying if a newly created k-cardinality tree, either by mutation or recombination or Local-Search, can be added to the current population. Two variants have been considered. In both cases one requirement is that the new solution should have an objective function (which is calculated by the Evaluate() method) better than the worst solution in the current population. In the first variant, we do not allow two solutions in

RestartPop()

begin
  repeat
  Pick a vertex $v \in NRL$;
  Apply Prim's algorithm starting at v to find a minimum spanning tree $ST(v)$ of G;
  Apply DynamicTree to $ST(v)$ to find a k-cardinality tree $T(v)$;
  Remove v from NRL
  until 25 or more different k-cardinality trees have been generated
  Return the best 25 k-cardinality trees found plus the best one from the previous population.
end;

Figure 6: RestartPop() based on DynamicPrimTree

the population to have the same objective function. If that is the case, we compare them both with the best solution in the population and we keep in the population the one that has less edges in common with the current best. In the second variant, we allow solutions with the same objective function value to be in the population, but we do not allow repetitions, so the solutions must have at least one edge different and that does not include a leaf.

2.2.2 The methods selectToMerge, selectToMutate, meetsPopConvCriteria, and the termination condition used

The **selectToMerge** method is responsible for choosing two parents to pass them to the Recombine() procedure. We considered two different procedures. In one of them we always choose the best individual in the population while the other is chosen at random with uniform probability. The **selectToMutate** chooses, with random uniform probability, a solution from the population to be the input of the Mutate() procedure. This implies that more than one mutation can be made to a particular solution. Note, however, that the **addInPopulation** method may actually prevent the resulting solution being included in the next population.

We consider that the population has lost diversity if after 13 $\log_2(13)$ generations the best solution in the population (i.e. the incumbent solution) has not changed. So in this case the **meetsPopConvCriteria** is very simple and just checks if this condition has occured. The memetic algorithm stops after having completed $13\, log_2(13)\, log_2(|E|)$ generations, where $|E|$ is the number of edges of the input graph G. The values of *#recombinations* and *#mutations* have been chosen to be 13 in both cases.

Ongoing work

We are currently implementing and testing the different MAs variants presented in this paper. The experimental results will be reported in a full version of the paper. In addition, current efforts are directed towards making the algorithm even simpler, for instance, by finding more efficient recombination algorithms having the same performance. Having in mind principles of generic programming and code reuse, we expect to deliver a metaheuristic that can be easily implemented using standard libraries for data structures and algorithms like LEDA.

Acknowledgements: This research was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and the CICYT project TIC1999-0754-C03 (MALLBA). P. Moscato acknowledges the finantial support from FAPESP, São Paulo, Brazil.

References

[BFM97] R. Borndörfer, C. Ferreira, and A. Martin. Matrix Decomposition by Branch-and-Cut. Technical Report SC-97-14, Konrad-Zuse-Zentrum für Informatiosntechnik, Berlin., 1997.

[BFM98] R. Borndörfer, C. Ferreira, and A. Martin. Decomposing Matrices Into Blocks. *SIAM Journal on Optimization*, 9(1):236–269, 1998.

[BFM99] L. Buriol, P. França, and P. Moscato, A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. Submitted to *Journal of Heuristics*, Oct. 1999.

[BX00a] M.J. Blesa and F. Xhafa. A C++ Implementation of a Skeleton for Tabu Search Method. Technical Report LSI-00-47-R, Dept. de LSI, UPC, 2000.

[BX00b] M.J. Blesa and F. Xhafa. A C++ Implementation of of Tabu Search for k-Cardinality Tree Problem Based on Generic Programming and Component Reuse. GCSE Young Researchers Workshop 2000 (Part of the Second International Symposium on Generative and Component-based Software Engineering) October 9-12, 2000. Erfurt, Germany.

[EFHM97] M. Ehrgott, J. Freitag, H. Hamacher, and F. Maffioli. Heuristics for the k-Cardinality Tree and Subgraph Problems. *Asia-Pacific J. of Op. Res.*, 14:87–114, 1997.

[FH92] L. Foulds and H.W. Hamacher. A New Integer Programming Approach to (Restricted) Facilities Layout Problems Allowing Flexible Facility Shapes. Technical Report 1992-3, Dept. of Management Science, University of Waikato, 1992.

[FHW97] L. Foulds, A. Hamacher, H.W.Philpott, and N. Wormald. On the Optimal Extraction of Ore from an Open-Cast Mine. Technical report, University of Auckland, New Zeland, 1997.

[FHW98] L. Foulds, H. Hamacher, and J. Wilson. Integer Programming Approaches to Facilities Layout Models with Forbidden Areas. *Annals of Operations Research*, 81:405–417, 1998.

[GH97] N. Garg and D. Hochbaum. An $O(\log k)$ Approximation Algorithm for the k-Minimum Spanning Tree Problem in the Plane. *Algorithmica*, 18:111–121, 1997.

[Glo86] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Op. Res.*, 5:533–549, 1986.

[HJ93] H.W. Hamacher and K. Jörnsten. Optimal Relinquishment According to the Norwegian Petrol Law: A Combinatorial Optimization Approach. Technical Report 7/93, Norwegian School of Economics and Business Administration, 1993. submitted to *Applied Mathematical Modelling*.

[HJM91] H.W. Hamacher, K. Jörnsten, and F. Maffioli. Weighted K-Cardinality Trees. Technical Report 91.023, Dept. di Elettronica, Politecnico di Milano, 1991.

[HTW97] A. Hertz, E. Taillard, and D. de Werra. *Local Search in Combinatorial Optimization.*, Chapter 5, pp. 121–136. Emile Aarts and Jan Karel Lenstra Ed., 1997. John Wiley & Sons Ltd.

[JL97] K. Jörnsten and A. Løkketangen. Tabu Search for Weighted k-Cardinality Trees. *Asia-Pacific J. of Op. Res.*, 14(2):9–26, 1997.

[M99] P. Moscato. Memetic Algorithms: A short introduction. Chapter 14 of *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover (Eds.), McGraw-Hill, pp. 219-234, 1999.

[TCK94] L. Trevisan, S. Cheung, and A. Kumar. Efficient Quorumcast Routing Algorithms. In *Proceedings of INFOCOM 94*, pages 840–847. IEEE Society Press, Los Alamitos, 1994.