Exact and Heuristic Approaches to the Maximum Capacity Representatives Problem

Italos Estilon da Silva de Souza[1], **Mauro Roberto Costa da Silva**[1], **Welverton Rodrigues da Silva**[1], **Rafael C. S. Schoeury**[1]

[1]Institute of Computing – University of Campinas (UNICAMP)
Caixa Postal 6176 – 13083-852 – Campinas – SP – Brazil

italosestilon@gmail.com, maurorcsc@gmail.com,
welvertonrodrigues@gmail.com, rafael@ic.unicamp.br

Abstract. *This paper approaches the problem of finding the system of representatives of a family of disjoint sets. To solve this problem, three methods were used: integer programming, branch-and-bound, and the BRKGA metaheuristic. We observed that, in randomly generated instances, the branch-and-bound algorithm was the best exact method but it was surpassed by BRKGA for large instances.*

1. Introduction

In this work, we address the problem of maximizing the sum of capacities of representatives of disjoint sets introduced by [Bellare 1993], who showed that this problem is NP-complete and gave some inapproximability results. Formally, the maximum capacity representatives problem (MCR) can be defined as follows: let $S_1, S_2, \ldots, S_k$ be disjoint sets. For any $i \neq j$, $x \in S_i$ and $y \in S_j$, let $c(x, y)$ be the nonnegative capacity between x and y. Let R be a set such that $\forall_{1 \leq i \leq k} |R \cap S_i| = 1$, we say that R is a system of representatives. The capacity C_R of a system of representatives R is given by $C_R = \sum_{x,y \in R} c(x, y)$. The problem is finding R that maximizes C_R.

One can think of an instance of this problem as an undirected graph in which the vertex set V is $\bigcup_{i=1}^{k} S_i$ and edge set E is $\{(u, v) \,|\, u \in S_i,\, v \in S_j, \text{ and } i \neq j\}$. The edge weights represent the capacity between the vertices and each S_i is an independent set. Note that this graph has the property of being k-partite complete.

Thus the problem of finding the system of representatives of maximum capacity is equivalent to the problem of finding a maximum edge-weighted clique in a k-partite complete graph. This is interesting because the maximum edge-weighted clique problem is widely studied. For simplicity, we will think about instances of the MCR problem as graphs and then, throughout this text, we will refer to sets S_i as parts, to capacities between elements as edge weights, and to elements of S_i as vertices.

We have not found the MCR problem to be studied experimentally in the literature, but we link to a variant problem, the so-called minimum distance representative (MinDR) [Blanco et al. 2014].

We approached the MCR problem with integer programming, branch-and-bound, and BRKGA, comparing these techniques.

2. Integer Programming

[Gouveia and Martins 2015] consider a version of the problem of maximum edge-weighted clique where there are edges with negative weight. They gave the formulation

below to this problem based on [Park et al. 1996]. Let $G = (V, E)$ be an undirected graph. For all $i \in V$, variable x_i equals 1 if i is in the clique, otherwise x_i equals 0. For all $i, j \in V$, variable y_{ij} equals 1 if edge (i, j) is in the clique, otherwise y_{ij} equals 0.

$$\text{maximize} \sum_{i,j \in E} c_{ij} y_{ij} \tag{1}$$

$$\text{subject to } y_{ij} \leq x_i,\ y_{ij} \leq x_j \qquad (i,j) \in E \tag{2}$$

$$x_i + x_j \leq y_{ij} + 1 \qquad (i,j) \in E \tag{3}$$

$$x_i + x_j \leq 1 \qquad (i,j) \notin E \tag{4}$$

$$x_i \in 0, 1 \qquad i \in V \tag{5}$$

$$y_{ij} \in 0, 1 \qquad (i,j) \in E \tag{6}$$

We propose to exchange constraint (4) by $\sum_{x_j \in S_i} x_j = 1$ for all S_i, provided that we already know the graph is k-partite complete and we know each part S_i. The formulation with constraint (4) may lead to infeasible solutions to our problem because it does not force each solution to have exactly one vertex from each part, although an optimal solution of the former formulation is also an optimal solution to our problem.

3. BRKGA

The biased random-key genetic algorithm (BRKGA) is an evolutionary metaheuristic for optimization problems. This metaheuristic works with a fixed-size population composed of p vectors (chromosomes) of randomly generated numbers in the interval [0, 1). The population is divided into groups, the so-called elite, non-elite and mutants. For each iteration, the best ones are kept (elite), new ones are generated by crossover (non-elite) and a small number of mutants are introduced into the population.

The BRKGA uses a decoder to calculate the value of a chromosome [Resende 2011]. We create a deterministic algorithm for decoding that takes as input a chromosome and associates with it a feasible solution to the MCR problem and outputs this solution value. Each chromosome is a real vector $\mathbf{x} \in [0, 1)^k$, where each locus is sequentially associated to a disjoint set. Algorithm 1 presents the decoder.

Algorithm 1 Decoder algorithm

```
1: function DECODER(x)
2:     R ← ∅
3:     for i ← 1, ..., k do
4:         j ← 1 + ⌊x_i · |S_i|⌋
5:         R ← R ∪ {x}, where x is j-th element of S_i
6:     return Σ_{x,y∈R} c(x, y)
```

In addition, we create a greedy constructive heuristic to introduce a non-random key chromosome in the initial population. The heuristic works as follows: for all $i = 1, 2, \ldots, k$, the locus i is equal to $(j - 1)/|S_i|$, where x is the j-th element of S_i such that $\sum_{y \in V \setminus S_i} c(x, y)$ is the greatest sum of capacities.

4. Branch-and-Bound

We create an initial feasible solution by choosing for each S_i with $i = 1, 2, \ldots, k$ an $x \in S_i$ that maximizes $\sum_{y \in V \setminus S_i} c(x, y)$. That is, for each set S_i we choose the vertex that have the greatest sum of edge weights.

At level l of the search tree, we choose which vertex of part S_l will be in the clique. Note that the search tree has depth k, the number of sets. Let σ be a solution under construction, that is, the algorithm is on level $l < k$, and let σ_i be the vertex of part i in σ. Let $R(\sigma) = \sum_{\sigma_i,\sigma_j \in \sigma} c(\sigma_i, \sigma_j)$ be the value of solution under construction σ. Let $\mathrm{opt}(\sigma)$ be the greatest value σ can have after choose vertices for the remaining levels $l' > l$. Let $\rho(\sigma)$ be the sum of weights of heaviest edges between part S_i and S_j for all $i, j > l'$ plus the weight of the edge between σ_i and S_j for all $i \leq l$ and $j > l$. Note that $\mathrm{opt}(\sigma) \leq \rho(\sigma)$, that is, the best solution that can be made is less or equal to the sum of the weight of the heaviest edge that can be in a solution between each pair of parts plus the value of the solution under construction. Algorithm 2 presents the branch-and-bound algorithm.

Algorithm 2 Branch-and-Bound algorithm

```
1: σ_max ← ∅
2: BB(∅, 1)
3: function BB(σ, l)
4:     if l = k then
5:         if R(σ) > R(σ_max) then σ_max ← σ
6:         return
7:     if ρ(σ) ≤ R(σ_max) then return
8:     for all u ∈ S_l in non-increasing order by total capacity do
9:         BB(σ ∪ {u}, l + 1)
```

5. Computational Experiments

We generated the instances used in the experiments randomly. There are three types of instances, the ones with part sizes and weights were generated using a uniform probability distribution ("uniform"), the ones with parts of equal sizes ("similar"), and the ones with weights generated using the normal probability distribution ("normal"). Part sizes were chosen from the range of 1 to a maximum size using a uniform probability distribution. Edge weights were chosen from the range of 1 to 100, for instances of type "uniform", and from the range of 1 to 50, for type "similar", using a uniform probability distribution. For instances of type "normal", edge weights were chosen using a normal probability distribution with mean 100 and standard deviation of 2. Instances may have 5, 10, 30 or 50 parts and we used three values for the maximum size of a part: 10, 30 and 50. We made all combinations of instance types, the number of parts and maximum size of a part.

We implement the branch-and-bound algorithm (BB) in C++ and the ILP model in Java using Gurobi 7.5.2. The BRKGA was implemented in C++, using the API described and proposed in [Toso and Resende 2015]. We set parameters of BRKGA as follows: size of population $p = 1000$, elite set fraction $p_e = 0.16$, fraction of population to be replaced by mutants $p_m = 0.08$, probability that offspring inherit an allele from elite parent $rhoe = 0.65$, number of independent populations $p_n = 3$ and the number of generations $gen = 300$. Then we execute all instances for each algorithm with 1 hour of timeout in a machine with Intel (R) Xeon (R) Silver 4114 CPU @ 2.20GHz, 32GB of memory and Linux 64bits.

6. Results and Conclusions

The approach with integer programming did not lead to good results, provided that the solver only solved six instances. In cases where the solver found solutions close to optimal the gap was still large indicating that the solver was having difficulty to show that the solutions found were optimal. Instances of type "similar" were harder to solve by the ILP solver and by the branch-and-bound algorithm by the fact instances of type "similar" have more edges and vertices than the others instances. Instances of type "normal" and "uniform" had no significant difference in the number of vertices and edges. The branch-and-bound algorithm performed in a similar way for instances of those types, provided that, the number of vertices showed to be more determinative than edge weights for this method. For the ILP solver, edge weights distribution was not a significant factor, although it solved more instances of type "uniform" where the variance of weights is greater. The branch-and-bound approach obtained better results than the ILP since the BB algorithm solved sixteen instances.

The BRKGA was executed 100 times for each instance. The approach with BRKGA found, in average, solutions very close to optimal for instances solved by the branch-and-bound algorithm with optimality proof. For larger instances, BRKGA found solutions better than the ones found by the branch-and-bound algorithm within very competitive time, taking, in average, less than 15 seconds per instance. BRKGA found solutions equal to or better than those found by the exact methods for 31 instances. We tried to use BRKGA solutions as an initial lower bound to the branch-and-bound algorithm, but it was not able to prove optimality for those solutions, although it did not find better solutions before reaching the time limit. We conclude that BRKGA is a good approach for this problem given that it was better than exact methods for large instances. Provided that the exact methods did not lead to good results, we plan to analyze the use of other heuristics approaches as future work.

This project was supported by the São Paulo Research Foundation (FAPESP) grants #2014/25892-4, #2015/11937-9, #2016/23552-7, #2016/01860-1 and #2017/23343-1; and the Brazilian National Council For Scientific and Technological Development (CNPq) grants #311499/2014-7, #425340/2016-3 and #304856/2017-7.

References

[Bellare 1993] Bellare, M. (1993). Interactive proofs and approximation: reductions from two provers in one round. In *Theory and Computing Systems, 1993., Proceedings of the 2nd Israel Symposium on the*, pages 266–274. IEEE.

[Blanco et al. 2014] Blanco, R., Boldi, P., and Marino, A. (2014). Entity-linking via graph-distance minimization. *ArXiv e-prints*.

[Gouveia and Martins 2015] Gouveia, L. and Martins, P. (2015). Solving the maximum edge-weight clique problem in sparse graphs with compact formulations. *EURO Journal on Computational Optimization*, 3(1):1–30.

[Park et al. 1996] Park, K., Lee, K., and Park, S. (1996). An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 95(3):671–682.

[Resende 2011] Resende, M. G. (2011). Introdução aos algoritmos genéticos de chaves aleatórias viciadas. *Simpósio Brasileiro de Pesquisa Operacional*, pages 3680–3691.

[Toso and Resende 2015] Toso, R. and Resende, M. (2015). A c++application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93.