Computers & Operations Research 51 (2014) 123–129

Contents lists available at ScienceDirect

Computers & Operations Research

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

Optimization procedures for the bipartite unconstrained 0-1 quadratic programming problem

Abraham Duarte [a], Manuel Laguna [b], Rafael Martí [c], Jesús Sánchez-Oro [a]

[a] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, Spain
[b] Leeds School of Business, University of Colorado at Boulder, Boulder, CO, USA
[c] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain

ARTICLE INFO

Available online 27 May 2014

Keywords:
Quadratic programming
Branch and bound
Heuristic search
Tabu search
Iterated local search

ABSTRACT

The bipartite unconstrained 0-1 quadratic programming problem (BQP) is a difficult combinatorial problem defined on a complete graph that consists of selecting a subgraph that maximizes the sum of the weights associated with the chosen vertices and the edges that connect them. The problem has appeared under several different names in the literature, including maximum weight induced subgraph, maximum weight biclique, matrix factorization and maximum cut on bipartite graphs. There are only two unpublished works (technical reports) where heuristic approaches are tested on BQP instances. Our goal is to combine straightforward search elements to balance diversification and intensification in both exact (branch and bound) and heuristic (iterated local search) frameworks. We perform a number of experiments to test individual search components and also to create new benchmarks when comparing against the state of the art, which the proposed procedure outperforms.

1. Introduction

The bipartite unconstrained 0-1 quadratic programming problem (BQP) consists of selecting a subgraph that maximizes the sum of the weights associated with the chosen vertices and the edges that connect them. The problem is defined on a complete bipartite graph $G=(V,E)$, with $I=\{1,2,\ldots,m\}$ representing the set of vertices in the left-hand side of the graph, $J=\{1,2,\ldots,n\}$ representing the set of vertices in the right-hand side of the graph, E representing the set of edges that connect the vertices in I with the vertices in J, and $V=I\cup J$. There is a weight c_v associated with each vertex $v\in V$. There is also a weight q_{ij} that corresponds to the edge connecting vertices $i\in I$ and $j\in J$.[1] The problem consists of selecting $S\subseteq V$ such that the following function is maximized

$$f(S)=\sum_{v\in S}c_v+\sum_{i,j\in S}q_{ij}$$

Fig. 1 shows an example with $I=\{a,b,c\}$, $J=\{w,x,y,z\}$ and the table of edge weights. We assume that all vertex weights are zero (i.e., $c_v=0\ \forall\ v\in V$). We point out that weights, either on the vertices or the edges, can be positive, negative or zero.

Consider a solution $S_1=\{a,w,x,z\}$. The objective function value of this solution is

$$f(S_1)=q_{(a,w)}+q_{(a,x)}+q_{(a,z)}=8-4+13=17$$

A better solution is obtained by making the following vertex selections: $S_2=\{b,c,w,y,z\}$. The objective function value of solution S_2 is

$$f(S_2)=q_{(b,w)}+q_{(b,y)}+q_{(b,z)}+q_{(c,w)}+q_{(c,y)}+q_{(c,z)}$$
$$=1-7+24-15+8+20=31$$

In this case, an increase in the number of vertices from solution S_1 to solution S_2 resulted in an increase in the objective function of 14 units (31–17=14). However, this is not necessarily true in all cases. For instance, selecting a and c on the left side and y and z on the right side results in an objective function value of 38. This solution has four vertices and is better than solution S_2 that has 5 vertices.

The BQP has been studied in the literature under different names: maximum weight induced subgraph [19], maximum weight biclique [2], matrix factorization [7] or maximum cut on bipartite graphs [1]. From the point of view of heuristic optimization, however, the BQP has been somewhat neglected. In particular, to the best of our knowledge, there exist two articles – currently available on line – that describe several heuristics

E-mail addresses: Abraham.Duarte@urjc.es (A. Duarte), laguna@colorado.edu (M. Laguna), Rafael.Marti@uv.es (R. Martí), jesus.sanchezoro@urjc.es (J. Sánchez-Oro).

[1] Throughout our descriptions, we will use i to denote vertices in the left-hand-side of the graph (i.e., vertices in I), j to denote vertices in the right-hand-side of the graph (i.e., in J), and v to denote vertices in either side.

http://dx.doi.org/10.1016/j.cor.2014.05.019
0305-0548/

Edge	Weight
(a,w)	+8
(a,x)	-4
(a,y)	-3
(a,z)	+13
(b,w)	+1
(b,x)	0
(b,y)	-7
(b,z)	+24
(c,w)	-15
(c,x)	-10
(c,y)	+8
(c,z)	+20

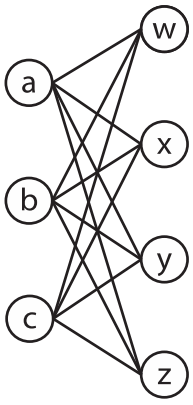

Fig. 1. BQP example.

for this problem [11]. This work develops 24 heuristics that are grouped into three families: fast-heuristics, slow-heuristics and row-merge heuristics.

In the reminder of the paper, we first introduce an exact method in the form of a branch-and-bound search. We describe this optimization procedure, as well as a lower bound, in Section 2. We then propose a heuristic procedure based on the iterated local search (ILS) methodology (Section 3). ILS has strong connections with the strategic oscillation originally proposed within tabu search [8]. Specifically, we propose two solution construction procedures (Section 3.1), two mechanisms to improve solutions via neighborhood searches (Section 3.2), and one perturbation strategy. Section 4 describes the computational experiments performed, and finally Section 5 presents the associated conclusions.

2. Branch-and-bound for the BQP

Branch and bound [13] is the process of systematically and exhaustively exploring the solution space by means of a search tree. See Wolsey [20] and Nemhauser and Wolsey [18] for classical references of this search strategy. Recent successful applications can be found in Martí et al. [17] and Martí et al. [16]. The search operates with bounds (lower and upper) on the optimal value of the objective function, a tree structure and exploration strategy. We obtain an initial lower bound (LB) with the heuristic procedures described in the following sections. This lower bound might change in the course of the B&B search and is used alongside an upper bound calculation in order to eliminate (i.e., prune) branches from further consideration.

We utilize a binary tree, where each node is associated with a vertex $i \in I$ in the graph. Each branch represents the decision of whether or not the vertex is included in the solution. Therefore, there are only two branches originating from each node in the tree, including the root node. At each node of the tree (except the root node), there is a set of vertices in I that have been selected, denoted by I'. We point out that it is not necessary to branch on vertices in J because given a selection of vertices in I the corresponding optimal subset of vertices in J can be trivially determined. This optimal selection of the vertices in J (denoted by J') generates a lower bound. In particular, a vertex $j \in J$ is selected if and only if

$$g(j) = c_j + \sum_{i \in I'} q_{ij} > 0$$

The set $S = I' \cup J'$ is a complete solution of the problem and therefore $f(S)$ can be used to update LB when $f(S) > LB$. While a single (the best) lower bound is maintained throughout the search, an upper bound (UB) is associated with each node. The upper bound is used to determine whether additional exploration rooted at the node is warranted. In particular, if for a given node it is determined that $UB \leq LB$, then there is no hope of finding a better solution by completing the sub-tree that is rooted at that node. Corresponding to each node of the tree, there is the set of selected vertices (i.e., I') and also the set I^U of unexplored vertices. The vertices $i \in I$ that belong to neither I' nor I^U are those that have been excluded from the solution by previous branching decisions. An upper bound associated with a node represented by (I', I^U) may be calculated as follows:

$$UB(I', I^U) = \sum_{i \in I'} c_i + \sum_{i \in I^U} \max(0, c_i) + \sum_{j \in J} \max\left(0, g(j) + \sum_{i \in I^U} \max(0, q_{ij}) + \sum_{i \in I'} q_{ij}\right)$$

The upper bound calculation is based on adding all the known weights (i.e., the weights associated with vertices that have been selected) and the strictly positive weights of the vertices that have not been explored. Then, we add the potential weight contribution of each vertex $j \in J$. Only strictly positive contributions are added to the upper bound calculation.

There are two standard techniques to explore a B&B tree: breadth-first and depth-first. Breadth-first generates wide trees and has demanding memory requirements. In most cases, B&B searches cannot be solely conducted on the basis of a breadth-first strategy due to computer memory limits. A depth-first approach attempts to find a leaf as fast as possible before moving to a different node. The memory requirements for depth-first are modest but the effectiveness of the approach is somewhat limited. The best B&B implementations use a mixed strategy that combines both approaches. In our mixed approach, we start with a breadth-first search until memory is exhausted, at which point, we switch to a depth-first exploration. Regardless of the exploring strategy, the direction $i \notin I'$ is always explored first.

3. Iterated local search for the BPQ

Iterated local search [14], usually referred to as ILS, is a metaheuristic based on a modification of local search or hill climbing methods for solving discrete optimization problems. Duarte et al. [3] and Lozano et al. [15] describe successful applications of this methodology. Algorithm 1 summarizes the ILS framework. An initial solution S_0 is generated that is immediately subjected to an improvement procedure (*LocalSearch*). The improved solution S_* becomes the starting point of the main ILS loop. This iterative loop consists of three main functions: *Perturb*, *LocalSearch* and *Accept*. *Perturb* typically employs random elements to change the current solution S_* to produce the perturbed solution S'. *LocalSearch* then attempts to find an improved solution S'_* and *Accept* implements the criteria by which the next current solution S_* is chosen. Both *Perturb* and *Accept* may use recorded history of the search to implement their strategies. For instance, it is possible to use frequency memory à la tabu search in order to bias perturbation mechanisms and acceptance criteria [4,5].

Although not explicit in Algorithm 1, the procedure keeps track of the best solution and returns it upon termination. The criteria within the *Accept* function create a balance between diversification and intensification. The criterion that encourages the most diversification is the one that always accepts S'_* and makes this solution the current solution (i.e.,