Quadratic Convex Reformulation : a Computational Study of the Graph Bisection Problem

Alain Billionnet[1], Sourour Elloumi[2], Marie-Christine Plateau[2]

15th March 2006

[1]Laboratoire CEDRIC, Institut d'Informatique d'Entreprise,
18 allée Jean Rostand, F-91025 Evry
billionnet@iie.cnam.fr
[2]Laboratoire CEDRIC, Conservatoire National des Arts et Métiers,
292 rue Saint Martin, F-75141 Paris
{elloumi, mc.plateau}@cnam.fr

Abstract

Given an undirected graph $G = (V, E)$, we consider the graph bisection problem, which consists in partitioning the nodes of G in two disjoined sets with p and $n - p$ nodes respectively such that the total weight of edges crossing between subsets is minimal. We apply QCR to it, a general method, presented in [4], which combines semidefinite programming (SDP) and Mixed Integer Quadratic Programming (MIQP). This method solves exactly general 0-1 quadratic programs (called (QP)) with linear constraints. It is composed of two phases : Phase I is devoted to reformulate (QP) into a new problem, equivalent to (QP), with a convex quadratic function; Phase II consists in submitting this new problem to a mixed-integer convex quadratic solver. Computational results reported in this paper show that the present approach is competitive with state-of-the-art methods for the graph bisection problem.

Keyword : Quadratic 0-1 programming, Convex quadratic reformulation, Semidefinite programming, Graph bisection problem, Experiments.

1 Introduction

Let $G = (V, E)$ be an undirected graph with n nodes $\{v_1, \ldots, v_n\}$ and a set of weigted edges E. The graph bisection problem consists in dividing the nodes of G into two sets V_1 and V_2 such that $\mid V_1 \mid = p$ and $\mid V_2 \mid = n - p$ and such that the total weights of edges that have end-points in different sets is minimal. This problem can be formulated as the following linearly constrained zero-one quadratic problem :

$$(BP) \ : \ \text{Min} \ \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left(x_i(1 - x_j) + (1 - x_i)x_j \right) : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

where c_{ij} is the weight of edge $[v_i, v_j]$. The binary variable x_i is equal to 1 if and only if the node v_i is in V_1.

This problem has many applications in various fields like scientific computing [10], physics [1], parallel algorithms [11]. It is known to be NP-hard [16]. So, many heuristics have been proposed ([2], [3], [7], [11], [19], [22], [25], [26]) as well as computations of lower bound ([5], [12], [13], [23], [15]).

Exact solution methods were studied in some papers. For the equicut problem (i.e. $p = \frac{n}{2}$), Roucairol and Hansen [27] suggested an approach based on the dualization of the size constraint. Brunetta, Conforti and Rinaldi [6] also proposed an exact solution method for the equicut problem that used branch-and-cut and linear programming relaxation. Ferreira and al. [14] described a similar approach for the more general node capacited graph partitioning problem while Johnson and al. [20] presented an approach based on column generation. For problems as (BP), Christofides and Brooker [8] proposed an algorithm based on the computation of a lower bound which is obtained by solving maximum flow problems. A parallel branch-and-bound algorithm was implemented by Clausen and Träff [9] while Michelon, Ripeau and Maculan [24] suggested a branch-and-bound method using an approximation of the feasible set by an ellipsoid.

All these exact solution methods were able to solve instances of arbitrary graphs with at most 60 nodes. An improvement was obtained by Karisch, Rendl and Clausen [21]. They describe an approach combining semidefinite programming and cutting plane. They can solve instances with 80-90 nodes and provide tight approximations for larger instances.

In this paper, we apply the Quadratic Convex Reformulation (QCR) method to the graph bisection problem. This exact solution method

was introduced in [4] for 0-1 quadratic programs (QP). It consists in reformulating (QP) into an equivalent 0-1 program with a convex quadratic objective function, followed by the use of a standard mixed integer quadratic programming solver. The reformulation is optimal in the sense that it provides the best possible continuous relaxation bound within a convexification framework.

This paper is organised as follows. Section 2 reviews the QCR method and shows how to apply it to the minimum bisection problem. Section 3 is devoted to computational experiments and comparisons with the results of [21]. Section 4 gives a conclusion.

2 Quadratic convex reformulation of the graph bisection problem

2.1 Recall of the general method QCR [4]

Consider the following linearly-constrained zero-one quadratic program :

$$(QP) \ : \ \text{Min} \ \left\{q(x) = x^t Q x + c^t x : Ax = b, \ A'x \leq b', \ x \in \{0,1\}^n\right\}$$

where c is an $n-$vector, b is an $m-$vector, b' is an $r-$real vector, Q is a symmetric $n \times n-$matrix, A is an $m \times n-$matrix and A' is an $r \times n-$matrix.

The QCR method consists in reformulating (QP) into an equivalent 0-1 program with a convex quadratic objective function $q_{\alpha,u}(x)$ depending on two parameters α and u. This new function is obtained by adding to $q(x)$ the two following functions, null on the feasible set, $q_1(x) = \sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right)$ and $q_2(x) = \sum_{i=1}^{n}\sum_{k=1}^{m} \alpha_{ki} x_i \left(\sum_{j=1}^{n} a_{kj} x_j - b_k\right)$. The following problem $(QP_{\alpha,u})$ is equivalent to (QP) :

$$(QP_{\alpha,u}) \ : \ \text{Min} \ \left\{q_{\alpha,u}(x) : Ax = b, \ A'x \leq b', \ x \in \{0,1\}^n\right\}$$

where

$$q_{\alpha,u}(x) = q(x) + q_1(x) + q_2(x)$$

The n-vector u and the $m \times n$-matrix α are determined in order to make $q_{\alpha,u}(x)$ convex and to maximize its minimal value over the relaxed domain

$\overline{X} = \{Ax = b, A'x \leq b', x \in [0,1]^n\}$. Optimal values of u and α are determined by solving a semidefinite relaxation of (QP).

In [4], this approach, applied to the densest k-subgraph problem, drastically improves state-of-the-art methods. In the following section, we apply it to the minimum graph bisection problem (BP).

2.2 Convex reformulation of the graph bisection problem

As mentionned in Section 1, the graph bisection problem can be formulated as follows :

$$(BP) \ : \ \text{Min} \ \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left(x_i(1-x_j) + (1-x_i)x_j\right) : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

which is equivalent to the following problem where, in the objective function, linear terms are separated from quadratic terms :

$$(BP) \ : \ \text{Min} \ \left\{ g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} - 2c_{ij}x_ix_j + \sum_{i=1}^{n} C_ix_i : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

where $C_i = \sum_{j=1}^{i-1} c_{ji} + \sum_{j=i+1}^{n} c_{ij}$. According to the general method recalled in Section 2.1, the reformulation of (BP) is :

$$(BP_{\alpha,u}) \ : \ \text{Min} \ \left\{ g_{\alpha,u}(x) : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

with :

$$g_{\alpha,u}(x) = g(x) + \sum_{i=1}^{n} u_i \left(x_i^2 - x_i\right) + \sum_{i=1}^{n} \alpha_i x_i \left(\sum_{j=1}^{n} x_j - p \right)$$

Since problem (BP) has just one constraint, α is a vector with n components. Let $(\underline{BP_{\alpha,u}})$ be the continuous relaxation of $(BP_{\alpha,u})$ i.e. :

$$(\underline{BP_{\alpha,u}}) \ : \ \text{Min} \ \left\{ g_{\alpha,u}(x) : \sum_{i=1}^{n} x_i = p, \ x \in [0,1]^n \right\}$$

The optimal parameters (α^*, u^*) can be obtained by solving the following semidefinite relaxation of (BP) whose optimal value is equal to the one of $(\underline{BP_{\alpha,u}})$:

$$(SDBP)\begin{cases} Min & \sum_{i=1}^{n}\sum_{j=i+1}^{n} -2c_{ij}X_{ij} + \sum_{i=1}^{n} C_i X_{ii} & & \\ s.t. & X_{ii} = x_i & i = 1,\ldots,n & (1) \\ & -px_i + \sum_{j=1}^{n} X_{ij} = 0 & i = 1,\ldots,n & (2) \\ & \sum_{j=1}^{n} x_i = p & & \\ & \begin{pmatrix} 1 & x^t \\ x & X \end{pmatrix} \succeq 0 & & \\ & x \in \mathbb{R}^n, X \in S_n & & \end{cases}$$

where S_n represents the set of $n \times n$ symmetric matrices.
u^* and α^* are precisely the optimal values of the dual variables associated to the equality constraints (1) and (2) respectively.

3 Computational results and comparison with existing methods

We use CPLEX9 to solve quadratic integer problems having a convex quadratic objective function and a set of linear constraints. We choose to solve semidefinite programs by using SB ([17], [18]), a software applying the spectral bundle method on eigenvalue optimization problems. All the numerical experiments have been carried out on a Pentium IV 2.2 GHz computer with 1 Go of RAM.
Note that one constraint is added to (BP) when $p = \frac{n}{2}$ in order to accelerate the branch-and-bound algorithm which consists in fixing to 1 the variable corresponding to a largest degree node.

3.1 Randomly generated unweighted graphs

We first apply QCR to randomly generated unweighted graphs. They are generated as follows : for a given density d and a couple of indices (i, j) such that $i < j$, we generate a random number ρ from $[0, 1]$. If $\rho > d$ then c_{ij} is set to 0, otherwise, c_{ij} is set to 1.
Tables 1 to 4 show the results for 5 graph sizes ($n = 40, 60, 80, 90, 100$), 3

densities ($d = 25\%, 50\%, 75\%$) and 3 p values ($p = \lfloor \frac{n}{8} \rfloor, \lfloor \frac{n}{4} \rfloor, \lfloor \frac{n}{2} \rfloor$). For each triplet (n, p, d), we solve 5 instances.

Legend of the tables :

- d, density of the graph.
- opt, value of the optimal or best known solution.
- CPU_{QCR}, total CPU time required by the MIQP solver of CPLEX9 to solve $(BP_{\alpha,u})$ and therefore (BP).
- CPU_{QCR} av is the average value of the CPU time for five instances. $best$ (resp. $worst$) CPU_{QCR}, the best (resp. worst) CPU time of all the five instances for a given density.
- $bound_{QCR}$, bound at the root of the branch-and-bound tree, i.e. optimal value of the continuous relaxation of $(BP_{\alpha,u})$ which is also the optimal value of $(SDBP)$.
- $gap_{QCR} = \frac{opt - bound_{QCR}}{opt} * 100$.
- $nodes$, number of nodes in the search tree.

Table 1 presents results for 4 values of n (i.e. $40, 60, 80, 90$). For each density, average values of CPU_{QCR} and gap_{QCR} are reported as well as the best and the worst CPU_{QCR} of the given five instances. Tables 2 to 4 (i.e. $n = 100$ nodes) present extensive results.

Table 1: Average results for randomly generated unweighted instances

n	p	$d(\%)$	CPU_{QCR} $av.$	$best\ CPU_{QCR}$	$worst\ CPU_{QCR}$	$gap_{QCR}(\%)$
40	$\lfloor \frac{n}{8} \rfloor$	25	0.04”	0.02”	0.07”	12.45
		50	0.07”	0.03”	0.12”	7.22
		75	0.08”	0.04”	0.14”	4.32
	$\lfloor \frac{n}{4} \rfloor$	25	0.15”	0.1”	0.29”	10.2
		50	0.25”	0.15”	0.45”	6.33
		75	0.36”	0.09”	1.01”	3.36
	$\lfloor \frac{n}{2} \rfloor$	25	0.54”	0.22”	1.37”	8.05
		50	0.64”	0.11”	0.85”	3.08
		75	0.53”	0.32”	1.02”	2.02
60	$\lfloor \frac{n}{8} \rfloor$	25	2.29”	0.5”	7.01”	13.68
		50	9.8”	1.8”	19.5”	5.62
		75	7.8”	1.8”	5.02”	2.8
	$\lfloor \frac{n}{4} \rfloor$	25	2”	1”	4”	9.86
		50	9.5”	1.6”	18”	5.64
		75	7.6”	1.7”	16.2”	2.82
	$\lfloor \frac{n}{2} \rfloor$	25	16.2”	4.5”	41”	7.16
		50	29”	9”	66”	3.72
		75	14.6”	8”	22”	1.78
80	$\lfloor \frac{n}{8} \rfloor$	25	5.7”	1.7”	8.6”	8.48
		50	3.8”	1.8”	8.7”	4.98
		75	14.8”	3.4”	36.8”	3.26
	$\lfloor \frac{n}{4} \rfloor$	25	2’18”	35.5”	7’	8.4
		50	4’04”	14.6”	11’05”	4.54
		75	5’09”	31.5”	9’46”	2.6
	$\lfloor \frac{n}{2} \rfloor$	25	24’21”	4’36”	50’38”	6.52
		50	10’51”	2’56”	24’13”	3.26
		75	6’49”	5’44”	10’30”	1.62
90	$\lfloor \frac{n}{8} \rfloor$	25	41.2”	1.5”	2’11”	10.04
		50	25.3”	9.11”	1’07”	5.14
		75	29.6”	10.38”	58.43”	3.08
	$\lfloor \frac{n}{4} \rfloor$	25	46’25”	16’31”	2h32’	8.76
		50	24’10”	46.6”	1h06’	3.48
		75	2h23’11”	27.1”	6h21’30”	2.56
	$\lfloor \frac{n}{2} \rfloor$	25	1h03’43”	16’58”	2h27’	5.92
		50	3h36’	9’48”	8h59’30”	3.14
		75	1h10’31”	17’28”	2h22’16”	1.6

Table 2: Results for randomly generated unweighted instances ($n = 100, p = \lfloor \frac{n}{8} \rfloor$)

$d(\%)$	num	opt	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$	$nodes$
25	1	178	46.8"	161.12	9.5	62548
	2	181	2'28"	161.33	10.9	192259
	3	182	29"	166.42	8.6	37599
	4	183	2'45"	163.31	10.7	211066
	5	189	20.7"	173.79	8	26134
50	1	420	1'4"	398.22	5.2	133224
	2	444	3'	418.02	5.8	226563
	3	426	1'24"	405.23	4.9	106652
	4	420	26.8"	397.83	5.3	33134
	5	424	4'48"	400.30	5.6	368080
75	1	695	51.3"	677.00	2.6	68699
	2	708	3'19"	686.15	3.1	274763
	3	723	3'24"	700.90	3.1	267130
	4	722	5'46"	698.38	3.3	474981
	5	719	2'	698.49	2.9	173249

Table 3: Results for randomly generated unweighted instances ($n = 100, p = \lfloor \frac{n}{4} \rfloor$)

$d(\%)$	num	opt	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$	$nodes$
25	1	335	4h36'32"	306.35	8.6	16548632
	2	326	27'13"	301.48	7.5	1613543
	3	340	1h17'	314.43	7.5	4473166
	4	331	2h08'36"	305.04	7.8	6651234
	5	356	2h27'	327.59	8	8589197
50	1	769	5h13'	734.73	4.5	17672160
	2	803	12h30'28"	761.04	5.2	43245303
	3	771	5h11'	738.89	4.2	18252528
	4	764	26'34"	733.02	4.1	1617654
	5	771	5h55'46"	738.40	4.2	20642738
75	1	1244	2'26"	1224.09	1.6	150053
	2	1271	2h07'22"	1242.62	2.2	6703433
	3	1301	4h34'45"	1271.36	2.3	16241715
	4	1292	1h23'11"	1265.53	2.0	4805338
	5	1292	42'55"	1266.43	2	2635526

Table 4: Results for randomly generated unweighted instances ($n = 100, p = \lfloor \frac{n}{2} \rfloor$)

$d(\%)$	num	opt	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$	$nodes$
25	1	459	13h51'24"	432.16	5.8	45887303
	2	452	14h22'47"	422.44	6.5	45334466
	3	473	29h30'03"	445.15	5.8	92752492
	4	451	7h06'19"	425.96	5.5	22557572
	5	487	20h04'	457.9	6	64184090
50	1	1039	14h51'11"	1005.47	3.2	47593207
	2	1067	1h42'31"	1036.35	2.8	4825501
	3	1044	43h41'31"	1010.35	3.2	140500675
	4	1036	2h15'51"	1008.25	2.6	6275713
	5	1048	49h39'47"	1012.82	3.3	157422406
75	1	1696	43h42'58"	1665.36	1.8	142751199
	2	1716	15h18'11"	1687.60	1.6	50102940
	3	1755	40h52'6"	1726.39	1.6	130657537
	4	1754	39h48'27"	1721.93	1.8	130988772
	5	1749	15h03'06"	1720.51	1.6	48732055

For all tables, the running times corresponding to the computation of (α^*, u^*) by the semidefinite programming solver SB are not reported : they are always less than one minute.

We give below some additional comments on Tables 1 to 4.

$n = 40, 60$ (Table 1)
For each instance, the optimal value is computed within about 2 seconds for $n = 40$ and within about 1 minute for $n = 60$.

$n = 80$ (Table 1)
For $p = \lfloor \frac{n}{8} \rfloor$, the CPU time is very small (always less than 37 seconds). For $p = \lfloor \frac{n}{4} \rfloor$, each instance is solved in less than 11 minutes. Worst results are obtained for $p = \lfloor \frac{n}{2} \rfloor$ and $d = 25\%$. However, all instances with $p = \lfloor \frac{n}{2} \rfloor$ are solved within 1 hour.

$n = 90$ (Table 1)
For $p = \lfloor \frac{n}{8} \rfloor$, the worst CPU time is about 2 minutes. For $p = \lfloor \frac{n}{4} \rfloor$, 11 instances out of 15 are solved in less than 1 hour. However, one instance (with $d = 75\%$) requires more than 6 hours. For $p = \lfloor \frac{n}{2} \rfloor$, 7 instances out of

15 are solved in less than 1 hour. The worst CPU time is equal to 9 hours for one instance with $d = 50\%$.

$n = 100$ (Tables 2 - 4)
For $p = \lfloor \frac{n}{8} \rfloor$, the worst CPU time is about 6 minutes. For $p = \lfloor \frac{n}{4} \rfloor$, the worst CPU time, equal to 12h30', is obtained for $d = 50\%$. However, the average of the CPU time for all the 15 instances nad for this value of p is 3h16'. For $p = \lfloor \frac{n}{2} \rfloor$, the worst CPU time is equal to about 50 hours for one instance with $d = 50\%$. In average, all the 15 instances are solved in 24 hours.

As a general remark, we can observe that, whatever the graph sizes are, the best gap values are obtained for largest densities. The average gap is equal to 9% for $d = 25\%$, 4% for $d = 50\%$ and 3% for $d = 75\%$.
Moreover, the longest running times are obtained for the graph equicut problem (i.e. $p = \lfloor \frac{n}{2} \rfloor$): intuitively, we can think that the partition size p increases the difficulty of the problem when it tends towards the value $\frac{n}{2}$ since the number of feasible solutions is equal to the number of combinations of p objects taken among n.

Comparison with a straightforward convexification

In order to solve (BP) by using an MIQP solver, there is an easy way to reformulate the objective function in order to make it convex. Let us present the reformulation when all c_{ij} are non negative. For all $x \in \{0,1\}^n$,

$$g(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij}(x_i - x_j)^2 = g_{conv}(x)$$

We have now the following new problem :

$$(BP_{conv}) \ : \ \text{Min} \ \left\{ g_{conv}(x) : \sum_{i=1}^{n} x_i = p, \ x \in \{0,1\}^n \right\}$$

whose objective function is convex. So, we can directly submit (BP_{conv}) to $CPLEX9$.
Note that the optimal value of the continuous relaxation of (BP_{conv}) is always equal to 0 since $x_i = \frac{p}{n}$ is a feasible solution of value 0. We test only two instances from Table 1 with $n = 40$, $p = \frac{n}{4}$ and two different densities ($d = 25\%$ and 75%). For the first (resp. second) instance, solving (BP_{conv}) needs 3'02" (resp. more than 2h30') while QCR takes 0.29" (resp. 0.45").

3.2 Comparison with Karisch, Rendl and Clausen (KRC) method

Recall of the KRC method [21]
In this section, we compare our method with the one of Karisch, Rendl and Clausen (KRC) who designed an exact solution method for the graph bisection problem. Their approach is a specific branch-and-bound algorithm based on semidefinite programming and polyhedral relaxations.

Comparison with the KRC method based on Brunetta, Conforti and Rinaldi instances
Karisch, Rendl and Clausen tested their approach on a library created by Brunetta, Conforti and Rinaldi (BCR) [6]. We choose to solve instances from it (*ftp://ftp.math.unipd.it/pub.Misc.equicut*) and compare our results with the ones of KRC. The instances correspond to $p = \lfloor \frac{n}{2} \rfloor$ and are divided into 4 classes :

- *Random Instances* (Table 5) where the density of the graphs is fixed before generating edges which integer weights are uniformly drawn from $[1, 10]$.
- *Toroidal Grid Instances* (Table 6) denoted by $'h \times kt'$. They represent a weighted $h \times k$ toroidal grid with edge weights uniformly drawn from $[1, 10]$, as previously. These graphs contains hk vertices and $2hk$ edges.
- *Mixed grid Instances* (Table 7) are complete graphs named $'h \times km'$. The edges of a planar $h \times k$ grid got weights uniformly drawn from [1,100], and all the other edges weights uniformly drawn from [1,10].
- *Instances with Negative Weights* (Table 8). They were generated in the same way as the instances of the first class except that the edges got weights from $[-10, -1] \cup [1, 10]$.

Note that our computer is about 15 times faster than the one of Karisch, Rendl and Clausen whose experiments were performed on a HP 9000/735.

Legend of Tables 5 - 8 :

- CPU_{KRC}, total CPU time required by the KRC method on a HP 9000/735.

Table 5: Equicut of randomly generated instances from the BCR-library

pb	n	$d(\%)$	*opt*	CPU_{KRC}	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$
v0.90	20	10	21	1"	0.01"	18.33	12.7
v0.00	20	100	401	1"	0.02"	392	2.2
t0.90	30	10	24	1"	0.01"	20.7	13.75
t0.50	30	50	397	22"	0.11"	370	6.8
t0.00	30	100	900	6"	0.09"	877.94	2.45
q0.90	40	10	63	4"	0.16"	50.72	19.4
q0.80	40	20	199	1'09"	0.54"	168.74	15.2
q0.30	40	70	1056	1'02"	0.98"	1016.35	17.9
q0.20	40	80	1238	25"	0.52"	1202.74	27
q0.10	40	90	1425	41"	0.64"	1387	2.67
q0.00	40	100	1606	4"	0.12"	1578.18	1.7
c0.90	50	10	122	10"	0.28"	102.06	16.3
c0.80	50	20	368	3'04"	2.66"	328.10	10.8
c0.70	50	30	603	4'02"	3.4"	555.9	7.8
c0.30	50	70	1658	2'44"	3.35"	1593.19	3.9
c0.10	50	90	2226	2'39"	2.03"	2166.95	2.65
c0.00	50	100	2520	2'20"	1.24"	2472.68	1.88
c2.90	52	10	123	12"	0.75"	96.58	21.4
c4.90	54	10	160	1'39"	1.28"	135.28	15.45
c6.90	56	10	177	30"	0.96"	149.41	15.6
c8.90	58	10	226	8'46"	23.61"	185.60	17.8
s0.90	60	10	238	4'57"	5.13"	200.82	15
	Average value			**1'35"**	**2"**		**11.4**

We solve all instances to optimality. For each class, the CPU time required to find the optimal solution is drastically improved compared with the results found by KRC : CPU_{QCR} can be up to 30 times smaller than CPU_{KRC} taking into account the difference of computers. More precisely, in average, all the running times are below 2" of *CPU* time and 37 instances out of 51 are solved within 1 second (49 within 5 seconds). The longest running time is about 4'18" for the largest negative instances ($n = 80$) and we don't improve the solution time of KRC method. For the other graphs, if we take into account the difference of platform, our method is between 3 and 16 times faster. The easiest classes of graphs are the mixed grid and toroidal grid instances whose maximum solution times are 3.69" and 0.81" respectively. In spite of very small *CPU* time, the gap obtained for toroidal

Table 6: Equicut of toroidal grid instances from the BCR-library

pb	n	$d(\%)$	*opt*	CPU_{KRC}	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$
4x5t	20	21	28	1”	0.01”	24.46	12.6
6x5t	30	14	31	3”	0.02”	22.65	26.9
8x5t	40	10	33	6”	0.09”	20.27	38.5
12x2t	42	10	9	5”	0.05”	3.008	66.5
23x2t	46	9	9	2’05”	0.2”	1.98	78
4x12t	48	9	24	17”	0.14”	10.99	54.2
5x10t	50	8	33	6”	0.25”	16.52	50
10x6t	60	7	42	5’50”	2.22”	19.52	53.5
7x10t	70	6	45	9’32”	3.69”	20.57	54.2
10x8t	80	5	43	15’44”	2.29”	23.51	47.7
	Average value			**3’22”**	**0.8”**		**48.21**

Table 7: Equicut of mixed grid instances from the BCR-library

pb	n	$d(\%)$	*opt*	CPU_{KRC}	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$
2x10m	20	100	118	1”	0.01”	112	5.1
6x5m	30	100	270	1”	0.03”	258.4	4.3
2x17m	34	100	316	29”	0.04”	294.98	6.6
10x4m	40	100	436	2”	0.04”	424	2.7
5x10m	50	100	670	2”	0.01”	654.126	2.4
4x13m	52	100	721	34”	0.23”	694.86	3.6
13x4m	52	100	721	34”	0.22”	694.82	3.6
9x6m	54	100	792	12”	0.48”	760.53	3.97
10x6m	60	100	954	8”	0.22”	932.29	2.3
10x7m	70	100	1288	14”	0.81”	1259.1	2.2
	Average value			**13.7”**	**0.2”**		**3.7**

instances is rather large due to the small densities.

Comparison with the KRC results obtained on their randomly generated instances

Finally, we apply our method on randomly generated instances, created by KRC. The graphs are unweighted random graphs with uniform edge probability $\frac{1}{2}$. Instances got a varying number of nodes from 36 to 84, and the p values are $\lfloor \frac{n}{2} \rfloor$, $\lfloor \frac{3n}{4} \rfloor$, $\lfloor \frac{7n}{12} \rfloor$ and $\lfloor \frac{13n}{24} + \frac{1}{2} \rfloor$. Table 9 present results.

Table 8: Equicut of instances with negative weights from the BCR-library

pb	n	$d(\%)$	*opt*	CPU_{KRC}	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$
t0.n.10	30	90	-301	7”	0.18”	-351.96	16.9
t0.n.00	30	100	-337	3”	0.09”	-382.63	11.9
q0.n.70	40	30	-298	50”	0.98”	-343.36	15.2
q0.n.50	40	40	-389	55”	0.37”	-435.46	11.9
q0.n.40	40	30	-450	6”	0.12”	-482.29	7.1
q0.n.00	40	100	-471	1’06”	0.56”	-544.37	15.58
c0.n.00	50	100	-829	4’17”	10.81”	-953.24	15.6
s0.n.80	60	20	-465	40”	3.34”	-513.65	10.46
o0.n.80	80	20	-690	31’48”	4’18”	-780.44	13.1
	Average value			**4’25”**	**30”**		**13.1**

Table 9: Randomly generated unweighted instances

graph	n	p	*opt*	CPU_{KRC}	CPU_{QCR}	$bound_{QCR}$	$gap_{QCR}(\%)$	$nodes_{QCR}$
ex36a	36	$\lfloor \frac{n}{2} \rfloor$	117	3”	0.22”	111.76	4.5	894
ex60a	60		367	5’	11.39”	354.45	3.4	26517
ex84a	84		742	2h02’55”	40’06”	716.36	3.5	2772955
ex36a	36	$\lfloor \frac{3n}{4} \rfloor$	85	31”	0.13”	78.41	7.8	584
ex60a	60		268	9’56”	8.95”	252.02	5.96	24569
ex84a	84		548	2h20’25”	12’24”	521.33	4.9	1193153
ex36a	36	$\lfloor \frac{7n}{12} \rfloor$	112	10”	0.21”	106.43	5	819
ex60a	60		351	5’38”	3.56”	340.64	2.9	8013
ex84a	84		721	9h16’51”	2h17’24”	691.56	4.08	9015554
ex36a	36	$\lfloor \frac{13n}{24} + \frac{1}{2} \rfloor$	114	3”	0.13”	108.88	4.5	794
ex60a	60		360	5’18”	8.72”	348.88	3.1	19349
ex84a	84		735	5h03’53”	1h52’48”	707.33	3.8	6437203
	Average value			**1h35’24”**	**25’16”**		**4.45**	

As Karisch et al. notified, randomly generated instances of this type constitute the most difficult classes. Let us remark that the p values are, for each instance, very close to $\frac{n}{2}$. So, it is the most difficult problem. All instances are solved to optimality. For largest graphs, the solution time for the different densities are very small for $n = 36$ (less than 1”) and $n = 60$ (less than 1’39”). For $n = 84$, several minutes and even hours are required (between 12’ and 2h17’). Note that, for all instances, the gap is rather small.

Now, if we compare our results with the ones of KRC (and obviously with the difference of platforms) our method is not always better. For instance, the optimal value of the graph with size of 84 and $p = \lfloor \frac{7n}{12} \rfloor$ is only found 4,5 times faster than the one of KRC whereas their computer is about 15 times slower. But we can say that Karisch et al. apply a specific method to the graph bisection problem whereas QCR is a very general approach which can be applied to many combinatorial optimization problems.

4 Conclusion

In this paper, we have presented an application of the QCR method to the graph bisection problem. This exact solution method deals with a reformulation of a minimization problem with a quadratic 0-1 function subject to linear constraints. The purpose of this transformation is to convexify the problem in order to solve it exactly by an MIQP solver. After good results obtained for the densest k-subraph problem [4], we have choosen to apply it to the graph bisection problem.

We solve about 300 instances of different classes : unweighted graphs, (positively and non positively) weighted graphs, grid instances. We consider diffrent values of densities and partition sizes. Our experiments show that the lower bound obtained by the gap is generally within 20% and decreases with graphs densities. For $d = 25\%$, the gap value is generally about 9%; for $d = 50\%$ about 4% and for $d = 75\%$ about 3%. The worst gap values are obtained for toroidal grids instances (due to the small graph densities).

The QCR approach solve bisection problems on general graphs with 90-100 nodes efficiently but note that, whatever the graph size, the equicut problem is the most difficult (especially for $n = 100$). So, we are competitive with state-of-the-art methods ([27], [6], [14], [20], [8],[9], [24]) who are limited to graphs with about 60 nodes.

Compared with the KRC method, we are competitve too: we are able to solve graphs with 90-100 nodes and we obtained tight relaxations for each class of problems. However, their solutions times are sometimes better than ours (around 7 instances out of 63) but they developed a very specific algorithm for the graph bisection problem whereas QCR is a very general method which can be apply to a lot of combinatorial optimization problems.

References

[1] F. Barahona, M. Grötschel, M. Jünger, and Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 1988.

[2] E.R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic and Discrete Mathematics*, 3:541–550, 1982.

[3] E.R. Barnes, A. Vanelli, and J.Q. Walker. A new heuristic for partitioning the nodes of a graph. *SIAM Journal on Discrete Mathematics*, 1:299–305, 1988.

[4] A. Billionnet, S. Elloumi, and M.C. Plateau. Qcr : Convex quadratic reformulation for exact solution of 0-1 quadratic programs. *Technical Report CEDRIC*, http://cedric.cnam.fr/PUBLIS/RC856.pdf, 2005.

[5] R.B. Boppana. Eigenvalues and graph bisection: an average case analysis. *Proceeding of the 28th annual symposium on computer sciences, IEEE London*, pages 280–285, 1987.

[6] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997.

[7] T.N. Bui and B.R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45:841–855, 1995.

[8] N. Christofides and P. Brooker. The optimal partitioning of graphs. *SIAM Journal on Applied Mathematics*, 30:55–69, 1976.

[9] J. Clausen and J. Larsson Träff. Implementation of parallel branch-and-bound algorithms - experiences with the graph partition problem. *Ann. Operations Research*, 33:331–349, 1991.

[10] C.C. de Souza, R. Keunings, L.A. Wolsey, and O. Zone. A new approach to minimising the frontwidth in finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 111:323–334, 1994.

[11] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. *In D.F. Hsu, A.L. Rosenberg, D. Sotteau, ,editors, Interconnection Networks and Mapping and Scheduling Parallel Computations. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 21:AMS, 57–73, 1995.

[12] W.E. Donath and A.J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Developments*, 66:420–425, 1994.

[13] J. Falkner, F. Rendl, and H. Wolkowicz. A computational study of graph partitioning. *Mathematical Programming*, 66:211–240, 1994.

[14] C.E. Ferreira, A. Martin, C.C de Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem : a computational study. *Mathematical Programming*, 81:229–256, 1998.

[15] I. Fischer, G. Gruber, F. Rendl, and R. Sotirov. Computational experiments with a bundle approach for semidefinite cutting plane relaxations of max-cut and equipartition. *Mathematical Programming, Ser. B*, 105:451–469, 2006.

[16] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theorical Computer Science*, 1:237–267, 1976.

[17] C. Helmberg. A c++ implementation of the spectral bundle method. *Manual version 1.1.1*, http://www.zib.de/helmberg/SBmethod, 2000.

[18] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, pages 673–696, 2000.

[19] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part1, graph partitioning. *Operations Research*, 37(6):865–892, 1989.

[20] E.L. Johnson, A. Mehrotra, and G.L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.

[21] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.

[22] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *the Bell System Technical Journal*, 49:291–307, 1970.

[23] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming, Ser. B*, 95:91–101, 2003.

[24] P. Michelon, S. Ripeau, and N. Maculan. Un algorithme pour la bipartition d'un graphe en sous-graphes de cardinalité fixée. *RAIRO Operations Research*, 35:401–415, 2001.

[25] H. Pirkul and E. Rolland. A lagrangian based heuristic for uniform graph partitioning. *Working paper, A. Gary Anderson Graduate School of Management, University of California, Riverside, USA*, 1996.

[26] E. Rolland, H. Pirkul, and F. Glover. Tabu search for graph partitioning. *Ann. Operations Research*, 63:209–232, 1997.

[27] C. Roucairol and P. Hansen. Problème de la bipartition minimale d'un graphe. *RAIRO*, 21:325–348, 1987.