8-2007

A Level-2 Reformulation–Linearization Technique Bound for the Quadratic Assignment Problem

Warren P. Adams

Monique Guignard
University of Pennsylvania

Peter M. Hahn
University of Pennsylvania

William L. Hightower

Recommended Citation

Adams, W. P., Guignard, M., Hahn, P. M., & Hightower, W. L. (2007). A Level-2 Reformulation–Linearization Technique Bound for the Quadratic Assignment Problem. *European Journal of Operational Research, 180* (3), 983-996. http://dx.doi.org/10.1016/j.ejor.2006.03.051

A Level-2 Reformulation–Linearization Technique Bound for the Quadratic Assignment Problem

Abstract
This paper studies polyhedral methods for the quadratic assignment problem. Bounds on the objective value are obtained using mixed 0–1 linear representations that result from a reformulation–linearization technique (rlt). The rlt provides different "levels" of representations that give increasing strength. Prior studies have shown that even the weakest level-1 form yields very tight bounds, which in turn lead to improved solution methodologies. This paper focuses on implementing level-2. We compare level-2 with level-1 and other bounding mechanisms, in terms of both overall strength and ease of computation. In so doing, we extend earlier work on level-1 by implementing a Lagrangian relaxation that exploits block-diagonal structure present in the constraints. The bounds are embedded within an enumerative algorithm to devise an exact solution strategy. Our computer results are notable, exhibiting a dramatic reduction in nodes examined in the enumerative phase, and allowing for the exact solution of large instances.

Keywords
combinatorial optimization, assignment, branch and bound, quadratic assignment problem, reformulation–linearization technique

Disciplines
Other Mathematics

A Level-3 Reformulation-Linearization Technique Based Bound for the Quadratic Assignment Problem

by

Peter M. Hahn[1]
Electrical and Systems Engineering, The University of Pennsylvania
Philadelphia, PA 19104-6315, USA
hahn@seas.upenn.edu

Yi-Rong Zhu
Electrical and Systems Engineering, The University of Pennsylvania
Philadelphia, PA 19104-6315, USA
yrzhu@seas.upenn.edu

Monique Guignard
OPIM Department, The University of Pennsylvania
Philadelphia, PA 19104-6340, USA
guignard@wharton.upenn.edu

William L. Hightower
Mathematics and Computer Science, High Point University
High Point, NC 27262, USA
bhightower@linus.highpoint.edu

[1] Corresponding author (Current address: 2127 Tryon Street, Philadelphia, PA 19146-1228, USA).

Abstract:

We apply the level-3 Reformulation Linearization Technique (RLT3) to the Quadratic Assignment Problem (QAP). We then present our experience in calculating lower bounds using an essentially new algorithm, based on this RLT3 formulation. This algorithm is not guaranteed to calculate the RLT3 lower bound exactly, but approximates it very closely and reaches it in some instances. For Nugent problem instances up to size 24, our RLT3-based lower bound calculation solves these problem instances exactly or serves to verify the optimal value. Calculating lower bounds for problems sizes larger than size 25 still presents a challenge due to the large memory needed to implement the RLT3 formulation. Our presentation emphasizes the steps taken to significantly conserve memory by using the numerous problem symmetries in the RLT3 formulation of the QAP.

Key words: Quadratic Assignment, QAP, Reformulation Linearization, RLT, dual ascent, exact solution.

Acknowledgements: This material is based upon work supported by the National Science Foundation under grant No. DMI-0400155. The authors are grateful to management and consulting staff of the San Diego Supercomputing Center for the invaluable computational resources and guidance they provided for this work. We thank Professor Matthew Saltzman of the Mathematics Department at Clemson University for his expert advice on computational optimization matters.

Calculation of large RLT3-based QAP lower bounds was made possible by advanced computational resources deployed and maintained by Clemson Computing and Information Technology and the Clemson University International Center for Automotive Research Center for Computational Mobility Systems. The authors acknowledge the kind and competent assistance of the staff at the Cyber Infrastructure Technology Integration group. We also thank J. Barr von Oehsen, Director, Computational Science, Cyberinfrastructure Technology Integration at Clemson University for his help in adapting our code to the Sun Fire E6900 server.

Notation

Entries of a matrix E of size mxnx…xp, indexed by i,j,…,k, are denoted $e_{ij\dots k}$. Conversely, given numbers $\bar{e}_{ijknpqgh}$, one can form a corresponding matrix $\bar{E}$ of appropriate size. Z(P) will denote the optimal value of optimization problem (P).

1. Introduction

The quadratic assignment problem (QAP) is known as one of the most interesting and challenging problems in combinatorial optimization. It finds applications in facility location, computer manufacturing, scheduling, building layout design, and process communications. The standard mathematical formulation of the QAP is

[QAP]

$$\min \left\{ \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N} c_{ijkn}x_{ij}x_{kn} : \boldsymbol{x}\in \boldsymbol{X}, \boldsymbol{x} \text{ binary} \right\}, \tag{1-a}$$

$$\text{where } \boldsymbol{X} \equiv \left\{ \boldsymbol{x} \geq 0 : \sum_{i=1}^{N} x_{ij} = 1, \forall\left(j=1,\dots,N\right); \sum_{j=1}^{N} x_{ij} = 1, \forall\left(i=1,\dots,N\right) \right\}. \tag{1-b}$$

The QAP optimizes a quadratic function over the set of permutation matrices $\boldsymbol{x}$. Notice there is no quadratic term $x_{ij}x_{kn}$ in the objective function when $k=i$ or $n=j$ since the constraints force $x_{ij}x_{kn}=x_{ij}$ if $k=i$ and $n=j$, and $x_{ij}x_{kn}=0$ otherwise.

The QAP is one of the most difficult NP-hard combinatorial optimization problems, and instances of size $N>30$ can usually not be solved in reasonable CPU time. In addition the majority of QAP test problems have a homogeneous objective function, and this contributes to their difficulty, as this tends to produce weak lower bounds. Recent developments have produced improved, that is, tighter, bounds. The new methodologies include the interior point bound by Resende et al. (1995), the level-1 RLT-based dual-ascent bound by Hahn and Grant (1998), the dual-based bound by Karisch et al. (1999), the convex quadratic programming bound by Anstreicher and Brixius (2001), the level-2 RLT interior point bound by Ramakrishnan et al. (2002), the SDP bound by Roupin (2004), the lift-and-project SDP bound by Burer and Vandenbussche (2006), the bundle method bound by Rendl and Sotirov (2007), and the Hahn-Hightower level-2 RLT-based dual-ascent bound by Adams et al. (2007). The tightest bounds are the lift-and-project SDP bound and the two level-2 RLT-based bounds. However, when taking speed and efficiency into consideration, the most competitive bounds are the level-1 RLT-based dual-ascent bound by Hahn and Grant (1998), the convex quadratic programming bound by Anstreicher and Brixius (2001), and the Hahn-Hightower level-2 RLT-based dual-ascent bound by Adams et al. (2007).

The reformulation-linearization technique (RLT) is a strategy developed by (Adams and Sherali 1986, 1990; Sherali and Adams 1990, 1994, Sherali and Adams 1998, 1999) for generating tight linear programming relaxations for discrete and continuous nonconvex problems. For mixed zero-one programs involving n binary variables, RLT establishes an n-

level hierarchy of relaxations spanning from the ordinary linear programming relaxation to the convex hull of feasible integer solutions. For a given $d \in \{1,...,n\}$, the level-*d* RLT, or simply, RLT*d*, constructs various polynomial factors of degree d consisting of the product of some d binary variables x_j or their complements $(1-x_j)$. The procedure essentially works via two steps. First it reformulates the problem by adding to the level-(*d*-1) RLT formulation at least some of the redundant nonlinear restrictions obtained by multiplying each of the defining constraints with the product factors. Then it linearizes each distinct nonlinear term by replacing it with a new continuous variable in both objective function and constraints, yielding a mixed zero-one linear representation in a higher dimensional space. The set of redundant constraints should be chosen to guarantee that the mixed-zero-one model is equivalent to the original model, i.e., that every new variable, given all added constraints, is in fact equal to the product it replaces. At each level *d* of the RLT hierarchy, i.e., RLT*d*, the resulting continuous relaxation is at least as tight as its previous level, with the highest n-th level representing the convex hull of the feasible region.

Our prior computational experience using first RLT1 and then RLT2 formulations for the QAP has indicated promising research directions. The corresponding continuous linear relaxations, problems $\overline{\text{RLT1}}$[2] and $\overline{\text{RLT2}}$, are increasingly large in size and highly degenerate. In order to solve these problems, Hahn and Grant (1998) and Adams et al. (2007) have presented a dual-ascent strategy that exploits the block-diagonal structure of constraints in the RLT1 and RLT2 forms, respectively. This strategy is a powerful extension of that found in Adams and Johnson (1994); it does not actually calculate either the RLT1 or RLT2 bounds, but it approximates them very closely and occasionally does reach those bounds exactly. The accomplishment here is the speed and efficiency of the computations.

Problem $\overline{\text{RLT2}}$, in particular, provides sharp lower bounds, as shown in Table 1 of Loiola et al. (2007), and consequently leads to very competitive exact solution approaches. A striking outcome, documented in Table 2 of Loiola et al. (2007), is the relatively few nodes considered in the binary search tree to verify optimality. This leads to marked success in solving difficult QAP instances of size $N \geq 24$ in record computational time. Based on this success, we turn attention in this paper to the level-3 form in order to get even tighter bounds, knowing that we will have to pay a price for the increased model size. The challenge is to take advantage of the additional strength without being hurt by the substantial increment in problem dimensions. This will require novel computational steps, better adapted to the much larger formulation size. We will first show that, as for level 2, the level-3 form can be handled via a Lagrangean approach to obtain a subproblem with block-diagonal structure. This time, however, we have many more dualized constraints and decomposable subproblem blocks. We will also need a more sophisticated approach for handling the nested structure, as well as the complicating constraints.

In the next section, we derive the level-3 formulation, focusing attention in Section 3 on deriving level-3 bounds from a Lagrangean dual approach. Section 4 explains the issues involved in programming the algorithm and describes the various approaches adopted to face them. In Section 5, we compare the strength and calculation speed of the new RLT3-based

[2] We use the notation $\overline{(P)}$ to denote the continuous relaxation of the mixed-integer programming problem (*P*).

bounds with those available from RLT2. Section 6 gives a brief summary of our conclusions and discusses ongoing research.

2. The RLT3 formulation of the QAP

The proposed RLT3 reformulation consists of the following steps. First multiply each of the $2N$ assignment constraints by each of the N^2 binary variables x_{ij} (this is an RLT1 step). [3] Multiply each of the $2N$ assignment constraints by each of the $N^2(N-1)^2$ products $x_{ij}x_{kn}$, $k \neq i$ and $n \neq j$, (this is an RLT2 step). Then, multiply each of the $2N$ assignment constraints by each of the $N^2(N-1)^2(N-2)^2$ products $x_{ij}x_{kn}x_{pq}$, $p \neq k \neq i$ and $q \neq n \neq j$, (this is an RLT3 step). Append all these restrictions. Express the various resulting products in the order $x_{ij}x_{kn}$, $x_{ij}x_{kn}x_{pq}$ and $x_{ij}x_{kn}x_{pq}x_{gh}$. Substitute $x_{ij} = x_{ij}^2$ wherever such or similar products appear. Remove all products $x_{ij}x_{kn}$ if $(k=i$ and $n \neq j)$ or $(k \neq i$ and $n = j)$ in quadratic expressions, all products $x_{ij}x_{kn}x_{pq}$ if $(p=i$ and $q \neq j)$, $(p=k$ and $q \neq n)$, $(p \neq i$ and $q = j)$ or $(p \neq k$ and $q = n)$ in cubic expressions, and all products $x_{ij}x_{kn}x_{pq}x_{gh}$ if $(g=i$ and $h \neq j)$, $(g=k$ and $h \neq n)$, $(g=p$ and $h \neq q)$, $(g \neq i$ and $h = j)$, $(g \neq k$ and $h = n)$ or $(g \neq p$ and $h = q)$ in biquadratic expressions, as they must be zero given the model constraints. In the end, we obtain a nonlinear model in the original binary variables x_{ij}.

The second step linearizes the model by introducing new continuous variables and imposes additional restrictions on these variables. Replace each occurrence of the product of two *x* variables by a single nonnegative continuous variable *y* (like in RLT1), whose quadruple index will consist of the two indices of the first *x* variable followed by those of the second *x* variable. For instance, for $k \neq i$ and $n \neq j$, $x_{ij}x_{kn}$ is replaced by $y_{ij\,kn}$. Similarly, like in RLT-2, every product of three *x* variables is replaced by a new, six-index, *z* variable. Finally every product of four variables *x* is replaced by a new eight-index *v* variable. Commutativity within the *x* products implies symmetry between the new variables, for instance for the *y* variables, $y_{ij\,kn} = y_{knij}\ \forall (i,j,k,n=1,...,N), k>i, n \neq j$, (see 2l below), and similarly for variables *z* and *v* (see 2h and 2d below).

The resulting RLT3 formulation of QAP is given below. Notice that the coefficients $d_{ij\,knpq}$ and $e_{ij\,knpqgh}$ found in the objective function are in general zero, however we keep them in the model as our RLT3-based lower bound code is also capable of calculating tight lower bounds for genuine cubic and biquadratic assignment problems.

---

[3] Notice that given that all constraints, original or generated, are equality constraints, one does not need to multiply also by terms containing (1- x_{ij}).

$$[\text{RLT3}] \quad \min \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N} c_{ijkn}y_{ijkn} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N} d_{ijknpq}z_{ijknpq}$$
$$+\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N}\sum_{\substack{g=1\\g\neq i,k,p}}^{N}\sum_{\substack{h=1\\h\neq j,n,q}}^{n} e_{ijknpqgh}v_{ijknpqgh} \tag{2a}$$

s.t. $$\sum_{\substack{g=1\\g\neq i,k,p}}^{N} v_{ijknpqgh} = z_{ijknpq} \qquad i,j,k,n,p,q,h=1,\ldots,N; h\neq q\neq n\neq j, p\neq k\neq i, \tag{2b}$$

$$\sum_{\substack{h=1\\h\neq j,n,q}}^{N} v_{ijknpqgh} = z_{ijknpq} \qquad i,j,k,n,p,q,g=1,\ldots,N; g\neq p\neq k\neq i, q\neq n\neq j, \tag{2c}$$

$$v_{ijknpqgh} = v_{knijpqgh} = \ldots = v_{ghpqijkn} = v_{ghpqkn}$$
$$i,j,k,n,p,q,g,h=1,\ldots,N; g>p>k>i, h\neq q\neq n\neq j, \tag{2d}$$

$$v_{ijknpqgh} \geq 0 \qquad i,j,k,n,p,q,g,h=1,\ldots,N; g\neq p\neq k\neq i, h\neq q\neq n\neq j, \tag{2e}$$

$$\sum_{\substack{p=1\\p\neq i,k}}^{N} z_{ijknpq} = y_{ijkn} \qquad i,j,k,n,q=1,\ldots,N; q\neq n\neq j, k\neq i, \tag{2f}$$

$$\sum_{\substack{q=1\\q\neq j,n}}^{N} z_{ijknpq} = y_{ijkn} \qquad i,j,k,n,p=1,\ldots,N; p\neq k\neq i, n\neq j, \tag{2g}$$

$$z_{ijknpq} = z_{knijpq} = \ldots = z_{pqijkn} = z_{pqkn} \qquad i,j,k,n,p,q=1,\ldots,N; p>k>i, q\neq n\neq j, \tag{2h}$$

$$z_{ijknpq} \geq 0 \qquad i,j,k,n,p,q=1,\ldots,N; p\neq k\neq i, q\neq n\neq j, \tag{2i}$$

$$\sum_{\substack{k=1\\k\neq i}}^{N} y_{ijkn} = x_{ij} \qquad i,j,n=1,\ldots,N; n\neq j, \tag{2j}$$

$$\sum_{\substack{n=1\\n\neq j}}^{N} y_{ijkn} = x_{ij} \qquad i,j,k=1,\ldots,N; k\neq i, \tag{2k}$$

$$y_{ijkn} = y_{knij} \qquad i,j,k,n=1,\ldots,N; k>i, n\neq j, \tag{2l}$$

$$y_{ijkn} \geq 0 \qquad i,j,k,n=1,\ldots,N; k\neq i, n\neq j, \tag{2m}$$

$\boldsymbol{x} \in \boldsymbol{X}, \boldsymbol{x}$ binary. (2n)

The resulting model embeds the nested, progressively larger, models QAP, RLT1, RLT2, RLT3.

3. Lagrangean relaxation of the RLT3 model

One can show that model RLT3 is equivalent to the QAP when the binary constraints on $\boldsymbol{x}$ are enforced, just as with RLT1 and RLT2. RLT1 implied $y_{ijkn} = x_{ij}x_{kn}$, RLT2 additionally implied $z_{ijknpq} = x_{ij}x_{kn}x_{pq}$, finally RLT3 additionally implies $v_{ijknpqgh} = x_{ij}x_{kn}x_{pq}x_{gh}$. With the binary constraints on $\boldsymbol{x}$ relaxed, given that $\overline{\text{RLT3}}$ imbeds both $\overline{\text{RLT2}}$ and $\overline{\text{RLT1}}$, the tightest lower bound of all three RLT models comes from $\overline{\text{RLT3}}$. The $\overline{\text{RLT3}}$ model, however, is considerably larger than $\overline{\text{QAP}}$, $\overline{\text{RLT1}}$ and $\overline{\text{RLT2}}$. It is also highly degenerate, because from all equality constraints of RLT3, only $2N$ constraints in x have a nonzero right-hand-side. The challenge is to extract tight bounds from this formulation without paying a heavy computational price. Fortunately, every dual feasible solution of $\overline{\text{RLT3}}$ provides a lower bound for QAP, thus our strategy is to quickly compute near-optimal dual solutions.

We could obtain a smaller formulation of $\overline{\text{RLT3}}$ via the substitution suggested by constraints (2d), (2h) and (2l) without affecting the bound. The remaining variables *v*, *z* and *y* would be $v_{ijknpqgh}(g > p > k > i, h \neq q \neq n \neq j)$, $z_{ijknpq}(p > k > i, q \neq n \neq j)$ and $y_{ijkn}(k > i, n \neq j)$, making constraints (2d), (2h) and (2l) unnecessary. Here instead we will exploit a block-diagonal structure present within the Lagrangean relaxation subproblems that result from dualizing these constraints. Let $\bar{b}_{ij}$, $\bar{c}_{ijkn}$, $\bar{d}_{ijknpq}$ and $\bar{e}_{ijknpqgh}$ denote the objective coefficients associated with x_{ij}, y_{ijkn}, z_{ijknpq} and $v_{ijknpqgh}$ respectively. The resulting Lagrangean relaxation model $\overline{\text{RLT3M}}$ is

$\left[\overline{\text{RLT3M}}\right]$

$$\min \begin{cases} \sum_{i=1}^{N}\sum_{j=1}^{N}\bar{b}_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\bar{c}_{ijkn}y_{ijkn} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N}\bar{d}_{ijknpq}z_{ijknpq} \\ +\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\substack{k=1\\k\neq i}}^{N}\sum_{\substack{n=1\\n\neq j}}^{N}\sum_{\substack{p=1\\p\neq i,k}}^{N}\sum_{\substack{q=1\\q\neq j,n}}^{N}\sum_{\substack{g=1\\g\neq i,k,p}}^{N}\sum_{\substack{h=1\\h\neq j,n,q}}^{n}\bar{e}_{ijknpqgh}v_{ijknpqgh} \\ \text{s.t.} \\ (2b),(2c),(2e),(2f),(2g),(2i),(2j),(2k),(2m), \boldsymbol{x} \in \boldsymbol{X} \end{cases} \tag{3}$$

It is this formulation that underlies the algorithm discussed in this paper. The proof that the Lagrangean relaxation of (3) produces a valid lower bound is similar to that found in Adams et al. (2007) for the RLT2 formulation, and is available in Section 6 of Zhu (2007) and in an online version of this paper, Hahn et al. (2008). An important result in Section 6.5 of Zhu (2007) is Theorem 6-2, which shows how to decompose $\overline{\text{RLT3M}}$ into one assignment problem of size

N, N^2 assignment problems of size $N-1$ of optimal values γ_{ij} and optimal solutions $\tilde{\boldsymbol{y}}$, $N^2(N-1)^2$ assignment problems of size $N-2$ of optimal values $\eta_{ij\,kn}$ and optimal solutions $\tilde{\boldsymbol{z}}$, and $N^2(N-1)^2(N-2)^2$ assignment problems of size $N-3$ of optimal values $\varphi_{ij\,knpq}$ and optimal solutions $\tilde{\boldsymbol{v}}$. We present next a dual-ascent procedure, similar to that employed in Adams et al. (2007) for Problem $\overline{\text{RLT2}}$, but much more difficult to implement efficiently because of the increased model size, which provides a monotone non-decreasing sequence of lower bounds for the QAP via Lagrangean multiplier adjustments for $\overline{\text{RLT3M}}$.

Dual Ascent Procedure

By adding or subtracting multiples of equality constraints to the expression of the objective function, one does not modify its value, but one can modify its coefficients with the ultimate goal of introducing a, hopefully large, constant term that will act as a lower bound as long as all other modified objective function coefficients are kept nonnegative. First constraints like (2b), (2f), (2j), the original assignments constraints in **X**, and so on, can be used to move parts of coeeficients "down" the line from *v* to *z*, then to *y*, then to *x,* then to the constant. Furthermore, symmetry constraints can help modify coefficients within the pool of variables they connect together. As a group, (2b), (2c) and (2d) are especially potent for extracting large amounts from the associated cost matrix $\bar{\mathbf{E}}$ and transferring them to cost matrix $\bar{\mathbf{D}}$. This observation also applies to constraints (2f), (2g) and (2h), which enhance the movement of costs from matrix $\bar{\mathbf{D}}$ to cost matrix $\bar{\mathbf{C}}$ and to constraints (2j), (2k) and (2l), which enhance the movement of costs from matrix $\bar{\mathbf{C}}$ to cost matrix $\bar{\mathbf{B}}$, and ultimately the original assignment constraints are moving costs from cost matrix $\bar{\mathbf{B}}$ to the constant that is the lower bound on Z(GAP).

Notice first that in the dual ascent procedure, in order to save space, we do not need to store the actual multiplier values, but only the adjusted coefficients of matrices $\bar{\mathbf{B}}$, $\bar{\mathbf{C}}$, $\bar{\mathbf{D}}$ and $\bar{\mathbf{E}}$.

Constraints (2d), (2h) and (2l) have an additional benefit, in that they are instrumental in reducing the memory requirement of the lower bounding algorithm. It is not necessary to provide separate memory locations for the twenty-four elements of $\bar{\mathbf{E}}$ that correspond to the equated $\bar{\mathbf{V}}$ elements in (2d). One memory location for the sum of these cost elements is sufficient. The same holds true for the six elements of $\bar{\mathbf{D}}$ that correspond to the equated $\bar{\mathbf{Z}}$ elements in (2h). The same also holds true for the two elements of $\bar{\mathbf{C}}$ that correspond to the equated $\bar{\mathbf{Y}}$ elements in (2l). In order to use this memory saving construct, it is necessary to provide maps so that the algorithm, when dealing with a specific element in one of the three cost matrices $\bar{\mathbf{C}}$, $\bar{\mathbf{D}}$ or $\bar{\mathbf{E}}$ can point to the summed costs, in order that the sum can be updated when any changes are made to individual cost element values.

It is important to understand the effect that summing cost coefficients in $\bar{\mathbf{E}}$, $\bar{\mathbf{D}}$ and $\bar{\mathbf{C}}$ has on constraints (2b), (2c), 2(f), 2(g), (2j) and (2k). Consider first constraints (2b) and (2c), which relate the $\bar{\mathbf{E}}$ cost coefficients to the $\bar{\mathbf{D}}$ cost coefficients. When one considers only summed and stored values of $\bar{\mathbf{E}}$ and $\bar{\mathbf{D}}$, a specific stored sum of $\bar{\mathbf{D}}$ elements communicates with just four stored sums of $\bar{\mathbf{E}}$ elements. Regarding constraints (2f) and (2g), a specific stored sum of $\bar{\mathbf{C}}$ elements communicates with just three stored sums of $\bar{\mathbf{D}}$ elements. And, for constraints (2j) and 2(k), a specific stored sum of $\bar{\mathbf{B}}$ communicates with only two stored sums of $\bar{\mathbf{C}}$. These facts play an important role in the steps of the algorithm. The 24-fold equalities of (2d) are maintained, in that all permutations of the four pairs of subscripts on the $\bar{\mathbf{E}}$ cost coefficients are represented.

Here are the steps.

1. Initialize (3) by assigning $\bar{e}_{ij\,knpqgh} = e_{ij\,knpqgh} = 0$ for $\forall(i,j,k,n,p,q,g,h)$ with $g \neq p \neq k \neq i$ and $h \neq q \neq n \neq j$, $\bar{d}_{ij\,knpq} = d_{ij\,knpq} = 0$ for $\forall(i,j,k,n,p,q)$ with $p \neq k \neq i$ and $q \neq n \neq j$, $\bar{c}_{ij\,kn} = c_{ij\,kn}$ for $\forall(i,j,k,n)$ with $k \neq i$ and $n \neq j$, and $\bar{b}_{ij} = b_{ij}$ for $\forall(i,j)$, where $e_{ij\,knpqgh}$, $d_{ij\,knpq}$, $c_{ij\,kn}$ and b_{ij} are objective coefficients taken from $\overline{\text{RLT3}}$. Set the initial lower bound $Z = 0$. Set the iteration counter to be 0. Keep in mind that $c_{ij\,kn}$ and c_{knij} are summed and stored in a <u>single</u> memory location.

2a. For each (i,j), distribute the coefficient $\bar{b}_{ij}$ among the $(N-1)^2$ coefficients $\bar{c}_{ij\,kn}$ for all $k \neq i$ and $n \neq j$ by increasing each such $\bar{c}_{ij\,kn}$ by $\bar{b}_{ij}/(N-1)$ and decreasing $\bar{b}_{ij}$ to 0. This is equivalent, for each (i,j), to adding $\bar{b}_{ij}/(N-1)$ times each of the $N-1$ equations $\sum_{n \neq j} y_{ij\,kn} - x_{ij} = 0$ for all $k \neq i$ found in (2k) to the objective of (3). Keep in mind that $\bar{c}_{ijkn}$ and $\bar{c}_{knij}$ are summed and stored in a <u>single</u> memory location.

2b. For each (i,j,k,n) with $i \neq j$ and $k \neq n$, distribute the updated coefficient $\bar{c}_{ij\,kn}$ among the $(N-2)^2$ coefficients $\bar{d}_{ij\,knpq}$ for all $p \neq i,k$ and $q \neq j,n$ by increasing each such $\bar{d}_{ij\,knpq}$ by $\bar{c}_{ij\,kn}/(N-2)$ and decreasing $\bar{c}_{ij\,kn}$ to 0. This is equivalent, for each (i,j,k,n) with $i \neq j$ and $k \neq n$, to adding $\bar{c}_{ij\,kn}/(N-2)$ times each of the $N-2$ equations $\sum_{q \neq j,n} z_{ij\,knpq} - y_{ij\,kn} = 0$ for all $p \neq i,k$ found in (2g) to the objective of (3). Keep in mind that $\bar{d}_{ijknpq}$, $\bar{d}_{ijpqkn}$, $\bar{d}_{pqijkn}$, $\bar{d}_{knijpq}$, $\bar{d}_{knpqij}$, and $\bar{d}_{pqknij}$ are summed and stored in a <u>single</u> memory location.

2c. For each (i,j,k,n,p,q) with $i \neq j$, $k \neq n$ and $p \neq q$, distribute the updated coefficient $\bar{d}_{ij\,knpq}$ among the $(N-3)^2$ coefficients $\bar{e}_{ij\,knpqgh}$ for all $g \neq i,k,p$ and $h \neq j,n,q$ by increasing each such $\bar{e}_{ij\,knpqgh}$ by $\bar{d}_{ij\,knpq}/(N-3)$ and decreasing $\bar{d}_{ij\,knpq}$ to 0. This is equivalent, for each (i,j,k,n,p,q) with $i \neq j$, $k \neq n$ and $p \neq q$, to adding $\bar{d}_{ij\,knpq}/(N-3)$ times each of the $N-3$ equations $\sum_{h \neq j,n,q} v_{ij\,knpqgh} - z_{ij\,knpq} = 0$ for all $g \neq i,k,p$ found in (2c) to the objective of (3). Keep in mind that the elements of $\bar{\mathbf{E}}$ that correspond to the equated $\bar{\mathbf{V}}$ elements in (2d) are summed and stored together in a single memory location

3. Use the aforementioned THEOREM 6-2 from Zhu (2007) to sequentially solve (3) as $N^2(N-1)^2(N-2)^2+N^2(N-1)^2+N^2+1$ assignment problems.

3a. Solve $N^2(N-1)^2(N-2)^2$ assignment problems of size $N-3$ to obtain $\tilde{\boldsymbol{v}}$ and the value $\varphi_{ij\,knpq}$, as follows: Sequentially consider all (i,j,k,n,p,q) with $p\neq k\neq i$ and $q\neq n\neq j$, beginning with those (i,j,k,n,p,q) for which $\bar{d}_{ij\,knpq}$ prior to step 2c was 0. For a selected (i,j,k,n,p,q), for each $g\neq i,k,p$ and $h\neq j,n,q$, assign to coefficient $\bar{e}_{ij\,knpqgh}$ a percentage of the stored value of $\bar{\mathbf{E}}$ that contains $\bar{e}_{ij\,knpqgh}$, and subtract that amount from the corresponding sum in storage. (Four experimentally determined percentage values are involved, as there are four opportunities in this step to access a given summed $\bar{\mathbf{E}}$ storage location.) Upon solving the assignment problem of the resulting size $N-3$ matrix, add the now modified $\bar{e}_{ij\,knpqgh}$ values for $g\neq i,k,p$ and $h\neq j,n,q$ to their corresponding storage locations and increase $\bar{d}_{ij\,knpq}$ by the solution value $\varphi_{ij\,knpq}$. Proceed through all such (i,j,k,n,p,q) indices where $p\neq k\neq i$ and $q\neq n\neq j$.

3b. Solve $N^2(N-1)^2$ assignment problems of size $N-2$ to obtain $\tilde{\boldsymbol{z}}$ and the value $\eta_{ij\,kn}$ as follows: Sequentially consider all (i,j,k,n) with $k\neq i$ and $n\neq j$, beginning with those (i,j,k,n) for which $\bar{c}_{ij\,kn}$ prior to step 2b was 0. For a selected (i,j,k,n), for each $p\neq i,k$ and $q\neq j,n$, assign to coefficient $\bar{d}_{ij\,knpq}$ a percentage of the sum of $\bar{d}_{ij\,knpq}$, $\bar{d}_{knij\,pq}$, $\bar{d}_{ij\,pqkn}$, $\bar{d}_{knpqij}$, $\bar{d}_{pqij\,kn}$, and $\bar{d}_{pqknij}$ in storage, and subtract that amount from the stored sum. (Three experimentally determined percentage values are involved, as there are three opportunities in this step to access a given summed $\bar{\mathbf{D}}$ storage location.) Upon solving the resulting size $N-2$ assignment problem, add the now modified $\bar{d}_{ij\,knpq}$ values for $p\neq i,k$ and $q\neq j,n$ to their corresponding storage locations and increase $\bar{c}_{ij\,kn}$ by the solution value $\eta_{ij\,kn}$. Proceed through all such (i,j,k,n) indices where $k\neq i$ and $n\neq j$.

3c. Solve N^2 assignment problems of size $N-1$ to obtain $\tilde{\boldsymbol{y}}$ and the value γ_{ij} as follows: Sequentially consider all (i,j), beginning with those (i,j) for which $\bar{b}_{ij}$ prior to step 2a was 0. For a selected (i,j), for each $k\neq i$ and $n\neq j$, assign to the coefficient $\bar{c}_{ij\,kn}$ a percentage of the sum of $\bar{c}_{ij\,kn}$ and $\bar{c}_{knij}$ in storage, and subtract that amount from the stored sum. (Two experimentally determined percentage values are involved, as there are two opportunities in this step to access a given summed $\bar{\mathbf{C}}$ storage location). Upon solving the resulting size $N-1$ assignment problem, add the now modified $\bar{c}_{ij\,kn}$ values for $k \neq i$ and $n \neq j$ to their corresponding storage locations and increase $\bar{b}_{ij}$ by the solution value γ_{ij}. Proceed through all such (i, j) indices.

3d. Solve one assignment problem of size N to obtain $\tilde{\boldsymbol{x}}$. Upon doing so, place the equality constraints of $\boldsymbol{X}$ into the objective function with the optimal dual multipliers, adjusting the value of $\bar{b}_{ij}$ and increase the lower bound Z by the solution value of this size N assignment problem. Proceed to step 4.

4. If the binary optimal solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}, \hat{\boldsymbol{v}})$ to (3) is feasible to RLT3, i.e., if it satisfies (2d), (2h) and (2l), stop with $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}, \hat{\boldsymbol{z}}, \hat{\boldsymbol{v}})$ optimal to problem QAP. If it is not feasible to RLT3, stop if some predetermined number of iterations has been performed. Otherwise, increase the iteration counter by 1 and return to step 2a.

The dual-ascent procedure produces a nondecreasing sequence of lower bounds since Step 1 is input with all variables having nonnegative reduced costs. An additional step, *simulated annealing*, which is not discussed above, is one that was important in achieving good performance on the earlier RLT1-based (Hahn and Grant, 1998) and also on the RLT2-based bound calculations. This simulated annealing step involves returning random percentages of the lower bound to the $\bar{b}_{ij}$ coefficients on each round prior to Step 2a of the dual ascent procedure by dividing the returned amount equally among the rows of the $\bar{\mathbf{B}}$ matrix. The random percentages follow an exponentially decreasing annealing schedule. Experimentation is done to optimize the selection of the exponential rate for the annealing schedule, which affects algorithm speed as well as the bound achieved. This simulated annealing step serves to shake the dual ascent bound calculation out of local optima and, in previous work, sped the ascent of the bound calculation so that tight lower bounds were achieved sooner.

4. Essential computer programming considerations

To demonstrate the potential of our approach, we coded in FORTRAN a dual ascent algorithm that calculates level-3 bounds for the QAP. Programming these lower bound calculations presented a enormous challenge. Even though similar programs had been written for the RLT1-based and RLT2-based lower bound calculations, the size of the program was about to grow beyond hope of reasonable implementation on a typical computer available on the campus of today's universities. Thus, additional planning and care was essential to assure its feasibility by minimizing RAM requirement and adhering to good programming practice in an attempt to keep bound calculations from requiring long runtimes, so that the bound computation algorithm would eventually be usable in a branch-and-bound environment.

There are two primary considerations in designing our lower bound calculation programs for RLT1, RLT2 and RLT3. The first is the method of storing the cost coefficients that are manipulated in the lower bound algorithm. The second is the method by which those cost coefficients are indexed so that they can be accessed rapidly in performing the algorithmic steps described at the end of Section 3. In writing codes for RLT1- and RLT2-based bounds, we have implemented two storage and indexing methods. In Method #1, complementary cost variable elements are summed and stored in a single memory location and a map is derived which assigns the summed cost element to its locations in the cost matrix. In Method #2, the individual cost matrix elements are stored separately and code is written to move costs freely between complementary elements. The trade-off between the two methods favors the first method for minimizing RAM requirement and the second method for calculation speed. For RLT1, neither method stood out as being preferred. The maps which were required to implement Method #1 somewhat detracted from the potential memory savings. And, the speedups promised by Method #2 were not dramatic. For RLT3 however, the fact that memory savings would be dominated by the $\overline{\mathbf{E}}$ matrix, in which each cost element has 23 complementary partners, made it clear that Method #1 was the way to go. This is born out clearly in Table 1, wherein the storage of three copies of the $\overline{\mathbf{E}}$ matrix for a size 25 problem with Method #1 requires over 45 GB of RAM. Had Method #2 been implemented, the RAM requirement for just matrix $\overline{\mathbf{E}}$ would have been over 1 TeraByte.

Table 1 lists the major arrays in our FORTRAN implementation of the RLT3-based lower bound algorithm. Only the primary, most important arrays are listed. Added at the bottom of the list is an estimate of the memory required by the remaining working arrays and variables plus the amount of memory for the executable code. The TOTAL in the last line is an estimate of the amount of memory required by the algorithm to calculate the lower bound for a size 25 problem, based on an extrapolation of the memory requirements of the RLT3-based lower bound calculations for the six smaller Nugent instances.

Table 1 – Matrix sizes in Bytes for the problem size 25 RLT3-based lower bound calculation.

Variable	Dimension	No. Bytes
$\overline{\mathbf{B}}$ matrix integer values	two	2,116
$\overline{\mathbf{C}}$ matrix integer values	one	722,516
Real equivalents of $\overline{\mathbf{C}}$ matrix values	one	722,516
$\overline{\mathbf{D}}$ matrix integer values	one	126,960,000
$\overline{\mathbf{E}}$ matrix integer values	one	15,362,160,000
Map for locating $\overline{\mathbf{C}}$ matrix values	two	4,590,036
$\overline{\mathbf{C}}$ matrix values reverse map*	two	4,335,096
Map for locating $\overline{\mathbf{D}}$ matrix values	two	381,600,000
Map for locating $\overline{\mathbf{E}}$ matrix values	two	61,575,600,000
Temporary sorting matrix for $\overline{\mathbf{C}}$ values	one	722,516
Temporary sorting matrix for $\overline{\mathbf{D}}$ values	one	126,960,000
Temporary sorting matrix for $\overline{\mathbf{E}}$ values	one	15,362,160,000
Counter for accesses to $\overline{\mathbf{C}}$ matrix values	one	722,516
Counter for accesses to $\overline{\mathbf{D}}$ matrix values	one	126,960,000
Counter for accesses to $\overline{\mathbf{E}}$ matrix values	one	15,362,160,000
Y (quadratic) variable decision matrix (=0 or =1)	one	722,516
Program and miscellaneous working arrays		33,767,243,000
TOTAL		142,204,342,828

* $\overline{\mathbf{C}}$ reverse map is used to propogate **Y** (0-1) decisions.

The memory required to deal with $\overline{\mathbf{B}}$, $\overline{\mathbf{C}}$, $\overline{\mathbf{D}}$ and $\overline{\mathbf{E}}$ are the governing users of RAM memory. Of these, clearly $\overline{\mathbf{E}}$ is dominant. The temporary arrays for $\overline{\mathbf{B}}$, $\overline{\mathbf{C}}$, $\overline{\mathbf{D}}$ and $\overline{\mathbf{E}}$ are needed in order to back off from variable states that were caused in the simulated annealing step, since we do not want to save states that give us a poorer lower bound as a result of trying to shake up the coefficients when we try to improve the bound but fail. The arrays for counting access to $\overline{\mathbf{C}}$, $\overline{\mathbf{D}}$ and $\overline{\mathbf{E}}$ are used continually in the lower bound computation process in order to keep track of which individual cost coefficient is being utilized in the algorithm.

5. Computational experience

Table 2 compares the performance of the RLT3-based lower bound with that of the RLT2-based lower bound algorithm (denoted RLT2) on several difficult problem instances from the QAPLIB web site [9]. The RLT2-based bound values in Table 2 are essentially the best lower bound values achieved for these instances by any other method (see column 12 in http://www.seas.upenn.edu/qaplib/lowerbound.html).

Table 2. The QAP RLT3 lower bounds.

Inst-ance	Opt-imum	RLT3 LB w/SA	RLT3 sec w/SA	RLT3 LB w/o SA	RLT3 sec w/o SA	RLT2 LB w/SA	RLT2 secs
Chr25a	3796	3795.57*	409,398	3758.52	235,058	3796†	1,502
Had16	3720	3718.11*	~15,000	3719.1*	1,263	3720†	1,438
Had18	5358	5357.67*	44,680	5357.0*	8,722	5358†	3,137
Had20	6922	6919.1	48,020	6920.0*	31,955	6922†	8,288
Nug12	578	577.15*	1,468	577.2*	86	578	266
Nug15	1150	1149.74*	16,671	1149.1*	829	1150	978
Nug18	1930	1930**	86,951	1928.8*	10,940	1905	14,180
Nug20	2570	2569.05*	242,982	2568.1*	77,021	2508†	31,003
Nug22	3596	3590.44	90,782	3594.04*	100,095	3511	26,643
Nug24	3488	3486.12*	676,573	Not avail.	Not avail.	3369	38,529
Nug25	3744	Not avail.	Not avail.	3723.9	5,647,594	3577†	38,460
Rou15	354210	354210**	951	354209.2*	895	354210†	232
Rou20	725520	725314.4	252,282	724792.0	265,535	699390†	39,828
Tai20a	703482	703482**	254,432	703405.2	329,725	675870†	40,445
Tai25a	1167256	Not avail.	Not avail.	1133716.3	2,480,946	1091165†	27,035

* Optimum verified by RLT3-based lower bound code
** Problem solved exactly by RLT3-based lower bound code
† Recently re-calculated result by RLT2-based lower bound code

Since the RLT2 based bound calculations were made several years ago, we re-calculated the RLT2-based lower bounds, using the most up-to-date version of our RLT2 lower bound code. In some cases, the new RLT2 lower bound calculations confirmed earlier experimental results, whereas in other cases, improved RLT2 lower bound values were reached. For each test in Table 2 we present the best lower bound achieved and the number of seconds required for the calculation on a single 733 MHz cpu of a Dell 7150 server.

RLT3-based lower bounds for problem sizes ≤ 20 were calculated on a single cpu of a Dell PowerEdge 7150 server with 20 GB of RAM. Lower bounds for problem sizes 22 and 24 were calculated on a single cpu of an IBM terascale machine with 128 GB of RAM at the San Diego SuperComputing Center. For problem size 25, lower bounds were calculated on a single cpu of a Sun Fire E6900 server with 384 GB shared memory at the Clemson University Computational Center for Mobility Systems.

The RLT3 FORTRAN bound calculations reported in Table 2 were made with and without the simulated annealing step that was so important in achieving good performance on the earlier RLT1-based bound calculations (Hahn and Grant, 1998). The simulated annealing step serves to shake the dual ascent bound calculation out of local optima and, in previous work, sped the ascent of the bound calculation so that tight lower bounds were achieved sooner. However, in the case of RLT3, simulated annealing occasionally slows, rather than speeds up the achievement of tight bounds.

The most important point to be made in reading Table 2 is the fact that in all but the Nug25, Tai25a and Rou20 test instances, the RLT3-based algorithm found lower bounds that reached sufficiently close to the optimal solution that the best known solution was confirmed as optimal. This was true even for the very difficult Nug24 instance. An optimum is verified by the RLT3-based lower bound code when the RLT3-based lower bound reaches a value higher than any possible feasible solution of value less than the optimum value. One may wonder why, in some cases, it is necessary to reach lower bounds within 2.0 of the best known value to assure verification of the optimal solution value. This is so because in those instances, due to symmetries in the flow matrix, the solution set has only even solution values.

With simulated annealing, the RLT3-based lower bound calculation actually solved three of the problem instances exactly. In those three instances, testing the quadratic and linear costs determined that objective function cost reductions resulted in a pattern of zeros that constituted a zero cost feasible solution. For this reason alone, we have decided to continue to use simulated annealing in all future tests of the RLT3-based lower bounding method.

Just because a lower bound is tight, does not mean that one can count on it to be useful in a branch-and-bound algorithm. The bound has to be calculated quickly. Fortunately, the dual ascent bounds that we have developed (based upon RLT formulations level-1, level-2 and level-3) are calculated iteratively. After only a small number of iterations, the bounds grow to a significant percentage of its final value. This is demonstrated in Figure 1, using the experimental results for the Nug22 lower bound calculated by our RLT3-based algorithm. The graph in this Figure shows the fraction of the optimum solution value that is reached by the RLT3-based lower bound as a function of runtime on the DataStar IBM computer at the San Diego Supercomputing Center. Excellent lower bounds are reached in only a few minutes.

This paper does not discuss a branch-and-bound algorithm using the RLT3-based lower bound. The code for the branch and bound enumeration exists, but takes so much more memory than its lower bound calculation portion that one could solve only problems of size 18 or smaller. It would be impossible to glean useful information from solving such small problems. The runtimes would have been exorbitant compared to those for the much simpler RLT1-based branch and bound solver. Consider the fact that RLT2-based branch-and-bound runs are slower than RLT1-based branch-and-bound runs for problem sizes less than or equal to 22. See Adams, et al. (2007).

As mentioned before, the number of variables grows dramatically with RLT level. The $\overline{\text{RLT2}}$ branch-and-bound solver code already runs into memory limits of the current generation of computers for problem instances larger than $N = 36$. Memory limits of machines available to researchers today make it difficult, if not impossible to calculate RLT3 lower bounds for problem instances larger than $N = 25$ using the current Fortran code. On the positive side, $\overline{\text{RLT3}}$ experiments have demonstrated promise for reducing the number of nodes that must be considered for proving optimality using branch-and-bound. Figure 2 below demonstrates the growth in random access memory (RAM) with problem instance size, required for RLT-3-based lower bound calculations. The linear extrapolation is based on data from the lower bound experiments on six Nugent instances reported in Table 6-1. The largest problem for which we are able to calculate the RLT3-based lower bound using the current FORTRAN code is size 25. This requires exactly 173 GBytes of RAM.

6. Conclusion

This paper presents a level-3 reformulation-linearization technique (RLT) formulation of the QAP and our RLT3-based dual ascent procedure for lower bound calculations. RLT techniques, while showing great promise, have to date received little investigation in terms of practical implementation. We hope that the insights into the implementation of this new algorithm will help other researchers to improve upon our methods. It is the goal of our future efforts to show that practical means can be devised to make these techniques useful, not only for solving the QAP, but for solving large classes of similarly difficult combinatorial optimization problems.

FIG. 1 NORMALIZED LOWER BOUND VS RUNTIME

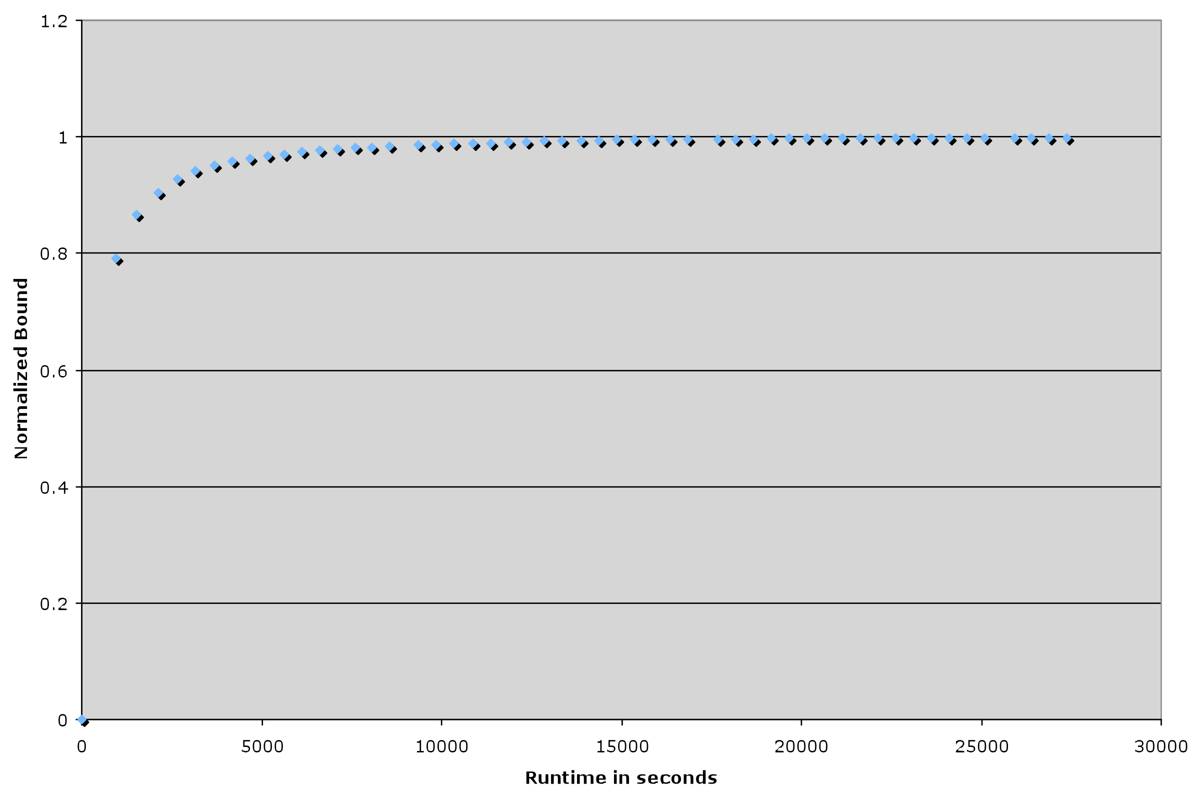
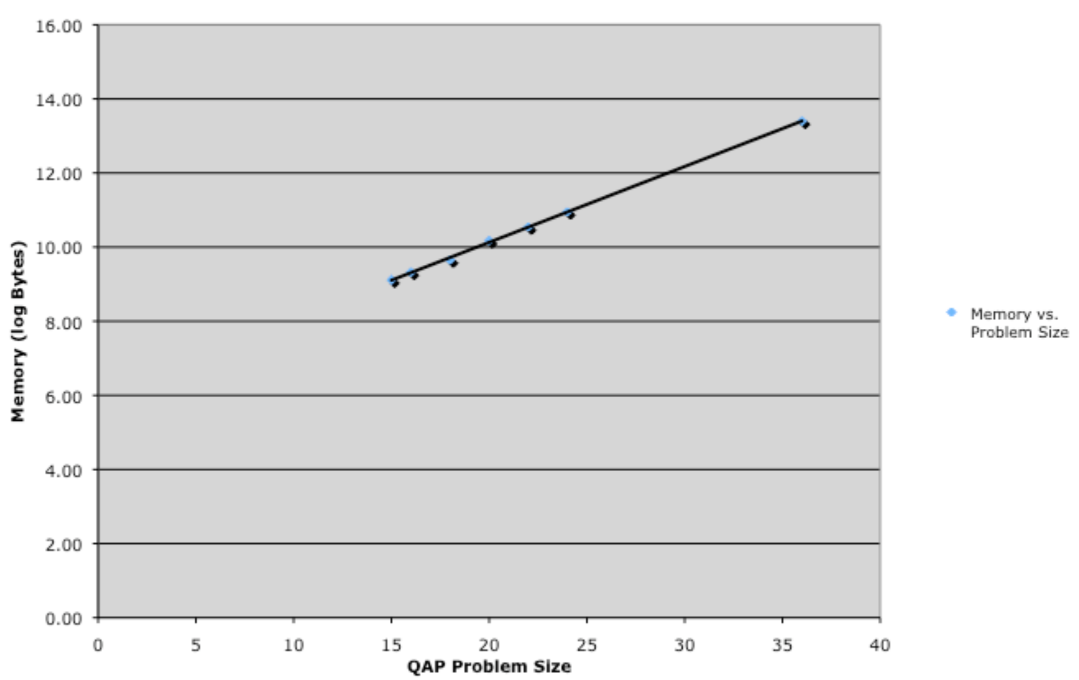


Figure 2. Memory for RLT-3 lower bound calculation



References

Adams, W.P., and T.A. Johnson, 1994, Improved linear programming-based lower bounds for the quadratic assignment problem, in: Pardalos, P.M., and H. Wolkowicz (Eds.), Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. **16,** AMS, Providence, RI, 43-75.

Adams, W.P., and H.D. Sherali, 1986, A tight linearization and an algorithm for zero-one quadratic programming problems, Management Science **32**, 1274-1290.

Adams, W.P., and H.D. Sherali, 1990, Linearization strategies for a class of zero-one mixed integer programming problems, Operations Research **38**, 217-226.

Adams, W.P., M. Guignard, P.M. Hahn, and W.L. Hightower, 2007, A level-2 reformulation-linearization technique bound for the quadratic assignment problem, European Journal of Operational Research **180**, 983-996.

Anstreicher, K.M., and N.W. Brixius, 2001, A new bound for the quadratic assignment problem based on convex quadratic programming, Mathematical Programming **89**, 341-357.

Burer, S., and D. Vandenbussche, 2006, Solving lift-and-project relaxations of binary integer programs, SIAM Journal on Optimization **16**, 726-750.

Gilmore, P.C., 1962, Optimal and suboptimal algorithms for the quadratic assignment problem, SIAM Journal on Applied Mathematics **10**, 305-313.

Hahn, P.M., and T.L. Grant, 1998, Lower bounds for the quadratic assignment problem based upon a dual formulation, Operations Research **46**, 912-922.

Hahn, P.M., Y-R. Zhu, M. Guignard and W.L. Hightower, 2008, A Level-3 Reformulation-linearization Technique Bound for the Quadratic Assignment Problem, Optimization OnLine, at http://www.optimization-online.org/DB_HTML/2008/09/2094.html.

Karisch, S.E., E. Çela, J. Clausen, and T. Espersen, 1999, A dual framework for lower bounds of the quadratic assignment problem based on linearization, Computing **63**, 351-403.

Lawler, E.L., 1963, The quadratic assignment problem, Management Science **9**, 586-599.

Loiola, E.M., N.M.M. de Abreu, P.O. Boaventura-Netto, P.M. Hahn, and T. Querido, 2007, A survey for the quadratic assignment problem, European Journal of Operational Research, **176**, 657-690.

Ramakrishnan, K.G., M.G.C. Resende, B. Ramachandran, and J.F. Pekny, 2002, Tight QAP bounds via linear programming, in: Pardalos, P.M., A. Migdalas, and R.E. Burkard (Eds.), Combinatorial and Global Optimization, World Scientific Publishing, Singapore, 297-303.

Rendl, F., and R. Sotirov, 2007, Bounds for the quadratic assignment problem using the bundle method, Mathematical Programming **109**, 505-524.

Resende, M.G.C., K.G. Ramakrishnan, and Z. Drezner, 1995, Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming, Operations Research **43**, 781-791.

Roupin, F., 2004, From linear to semidefinite programming: an algorithm to obtain semidefinite relaxations for bivalent quadratic problems, Journal of Combinatorial Optimization **8**, 469-493.

Sahni, S., and T. Gonzalez, 1976, P-complete approximation problems, Journal of the Association for Computing Machinery **23**, 555-565.

Sherali, H.D., and W.P. Adams, 1990, A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems, SIAM Journal on Discrete Mathematics **3**, 411-430.

Sherali, H.D., and W.P. Adams, 1994, A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems, Discrete Applied Mathematics **52**, 83-106.

Sherali, H.D., and W.P. Adams, 1998, Reformulation-linearization techniques for discrete optimization problems, in: Du, D-Z. and P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Vol. **1**, Kluwer Academic Publishers, Dordrecht, 479-532.

Sherali, H.D., and W.P. Adams, 1999, A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Y-R Zhu, 2007, "Recent advances and challenges in the quadratic assignment and related problems," Ph.D. dissertation, ESE Department, University of Pennsylvania, August 2007.