Submitted to *Operations Research*
manuscript OPRE-2011-06-292

Three ideas for the Quadratic Assignment Problem

Matteo Fischetti, Michele Monaci, Domenico Salvagnin
Department of Information Engineering, University of Padova, Via Gradenigo 6/B, 35121 Padova, Italy,
matteo.fischetti@unipd.it michele.monaci@unipd.it domenico.salvagnin@unipd.it

We address the exact solution of the famous `esc` instances of the quadratic assignment problem. These are extremely hard instances that remained unsolved—even allowing for a tremendous computing power—by using all previous techniques from the literature. During this challenging task we found that three ideas were particularly useful, and qualified as a breakthrough for our approach. The present paper is about describing these ideas and their impact in solving `esc` instances. Our method was able to solve, in a matter of seconds or minutes on a single PC, all easy cases (all `esc16*` plus `esc32e` and `esc32g`). The three previously-unsolved `esc32c`, `esc32d` and `esc64a` were solved in less than half an hour, in total, on a single PC. We also report the solution in about 5 hours of the previously-unsolved `tai64c`. By using a facility-flow splitting procedure, we were finally able to solve to proven optimality, for the first time, both `esc32h` (in about 2 hours) as well as "the big fish" `esc128` (to our great surprise, the solution of the latter required just a few seconds on a single PC).

Key words: integer programming, quadratic assignment problem
History: Submitted: 11 June 2011

In memory of Jens Clausen

1. Introduction

The NP-hard (and notoriously very difficult in practice) *Quadratic Assignment Problem* (QAP), in its Koopmans and Beckmann form [Koopmans and Beckmann(1957)], can be sketched as follows; see, e.g., [Cela(1998), Burkard et al.(2009)Burkard, Dell'Amico, and Martello] for details.

We are given a complete directed graph $G = (V, A)$ with n nodes n^2 arcs along with a set of n facilities to be assigned to its nodes. In what follows, indices $i, j \in V$ always correspond to nodes, indices $f, g \in N := \{1, \cdots, n\}$ to facilities, $b_{ij} \geq 0$ is a given (directed) *distance* from node i to node j, $a_{fg} \geq 0$ is a given required *flow* from facility f to facility g, and c_{if} is a given fixed cost for assigning facility f to node i. By using binary variables $y_{if} = 1$ iff facility f is assigned to node i, QAP can be stated as the following quadratic 0-1 problem:

$$\min \ \textstyle\sum_{i\in V}\sum_{f\in N}\sum_{j\in V}\sum_{g\in N} a_{fg}\, b_{ij}\, y_{if}\, y_{jg} + \sum_{i\in V}\sum_{f\in N} c_{if} y_{if} \tag{1}$$

$$\textstyle\sum_{i\in V} y_{if} = 1 \quad \forall f \in N \tag{2}$$

$$\textstyle\sum_{f\in N} y_{if} = 1 \quad \forall i \in V \tag{3}$$

$$y_{if} \in \{0,1\} \quad \forall i \in V, f \in N \tag{4}$$

Notice that the quadratic part of the objective function may hide some linear terms in the products $a_{ff} b_{ii} y_{if} y_{if}$ (as $y_{if}^2 = y_{if}$). Therefore in the following we will always assume, without loss of generality, $a_{ff} = 0$ and $b_{ii} = 0$ for all pairs (i, f), after the update $c_{if} := c_{if} + a_{ff} b_{ii}$.

It is well known that constraint matrix (2)-(3) is totally unimodular, so optimizing any *linear* objective function over QAP feasible set is just an easy LP problem solvable in $O(n^3)$ in the worst case, known as the *Linear Assignment Problem* (LAP) [Burkard et al.(2009)Burkard, Dell'Amico, and Martello].

In spite of its simple definition, QAP is among the most difficult optimization problems arising in practice, and its study attracted a large amount of research. A QAP feature that challenged us is that a collection of (apparently very small) test-cases is available in the QAPLIB [Burkard et al.(1991)Burkard, Karisch, and Rendl], that cannot be solved by the current state of the art even by allowing for tremendous computing power. E.g., an instance with just $n = 30$ such as `tho30` was solved only recently on a distributed grid [Anstreicher et al.(2002)Anstreicher, Brixius, Goux, and Linderoth] and required the equivalent of 8,997 days of computation—while many similar instances are still unsolved nowadays.

Among the difficult cases, we concentrated on the so-called `esc` instances introduced in [Eschermann and Wunderlich(1990)]. As reported in QAPLIB, instances `esc32a`, `esc32b`, `esc32c`, `esc32d`, `esc32h`, `esc64a`, and `esc128` are still unsolved by using any published method [Clausen and Perregaard(1997)]. It is fair however to mention that the "What's new" section of the QAPLIB webpage `http://www.seas.upenn.edu/qaplib/` states that in January 2011 Axel Nyberg and Tapio Westerlund at Abo Akademi University in Finland announced the solution of `esc32a`, `esc32c`, `esc32d`, `esc64a`, and `tai64c`. Instance `esc32c` took 9 hours on a single PC using the commercial solver Gurobi 3.0 (default settings), whereas `esc32d` took about 35 hours. As far as we know, however, no paper describing this method is available nor circulated in any way, so no comparison with our own approach can be made.

During our research we tested a number of ideas, and found that three of them were particularly useful, and qualified as a breakthrough for our approach. This paper is about describing these ideas and their effect in solving hard `esc` instances.

Our method was able to solve, in a matter of seconds or minutes on a single PC, all the easy cases (all `esc16*` plus `esc32e` and `esc32g`). The three previously-unsolved `esc32c`, `esc32d` and `esc64a` were solved in less than half an hour, in total, on a single quad-core PC. We also report the solution of the previously-unsolved `tai64c` in about 5 hours, on the same PC.

By using a facility-flow splitting procedure, we also succeeded in solving `esc128` to proven optimality—to our great surprise, this task took just a few seconds on our quad-core PC. According to QAPLIB, `esc128` is the largest QAP instance ever solved by an exact method. Instance `esc32h` was solved as well, and took about 2 hours on the same hardware, whereas for `esc32a` a very tight lower bound of 128 (out of 130) was proved in a matter of few minutes—the previous best-known bound was 103.

The paper is organized as follows. We describe the three main steps that proved to be crucial for the success for our approach, namely: exploiting symmetries (Section 2), designing and solving a suitable Mixed-Integer Linear Programming (MILP) formulation (Section 3), and exploiting a facility-flow splitting scheme to solve the hardest cases (Section 4). Some conclusions are finally drawn in Section 5.

An early version of the present paper has been presented at the CPAIOR'11 meeting held in Berlin, May 2011 [Fischetti et al.(2011)Fischetti, Monaci, and Salvagnin].

2. Step One: Exploiting symmetry

In his survey [Anstreicher(2003)], Anstreicher observed that "a careful consideration of the structure present in the larger `esc` instances is likely to be important in attempts to solve these problems to optimality". Indeed, `esc` instances are known to be highly symmetrical, and important attempts to deal with this property have been made in the literature, including those in [Kaibel(1998)] and [de Klerk and Sotirov(2010)].

Our approach was to study the symmetry of the QAP instance at hand—rather than that of its formulation–by addressing the symmetry groups induced by the flow and distance matrices a and b separately.

2.1. Symmetry in the flow matrix

A main source of symmetry for `esc` instances comes from the presence of what we call *clone* facilities—borrowing a concept introduced in [Balas and Fischetti(1993)] for the traveling salesman problem.

Definition 1 *Assume w.l.o.g. $a_{ff} = b_{ii} = 0$ for all i, f. We say that two distinct facilities f and g are* clones *(w.r.t. matrices a and c) if*

- $a_{fg} = a_{gf}$.
- $a_{fh} = a_{gh}$ *and* $a_{hf} = a_{hg}$ *for all* $h \in N \setminus \{f, g\}$;
- $c_{if} = c_{ig}$ *for all* $i \in V$.

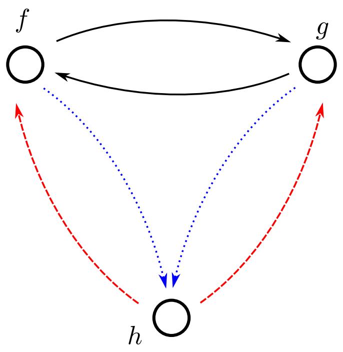

Figure 1 Graphical illustration of the first two clone conditions.

Figure 1 gives an illustration of the first two conditions.

It is self evident that the presence of clone facilities induces a symmetry in the problem, in the sense that, for a given clone pair (f, g) and for each $i \in V$, the two variables x_{if} and x_{ig} can be swapped without changing solution cost or feasibility. Of course, one could cope with clone symmetry by using a modified branching strategy such as isomorphism pruning [Margot(2002)] or orbital branching [Ostrowski et al.(2011)Ostrowski, Linderoth, Rossi, and Smriglio], but can one actually *take advantage* of symmetry?

To answer the question above, we observe that the clone property is an equivalence relation that partitions the facility set N into m (say) equivalence classes $C_1, \cdots, C_m$ that we call *clone clusters*, each of which plays the role of a different *facility type*. In what follows, we will always use indexes $u, v \in M := \{1, \cdots, m\}$ to denote facility types, and notation $\theta(f)$ to identify the type of facility $f \in N$, i.e., $f \in C_{\theta(f)}$. We also define a shrunken $m \times m$ flow matrix $\tilde{a}$ whose entries $\tilde{a}_{uv}$ define the flow to be sent from facility type u to facility type v, $\tilde{a}_{uu}$ being the (typically nonzero) required flow between any two distinct clones in C_u. Analogously, we define a shrunken $n \times m$ cost matrix $\tilde{c}$ whose entries $\tilde{c}_{iu}$ give the cost of assigning node i to facility type u. By definition,

$$\tilde{a}_{\theta(f),\theta(g)} = a_{fg} \quad \forall f, g \in N,\, f \neq g \tag{5}$$

$$\tilde{c}_{i,\theta(f)} = c_{i,f} \quad \forall i \in V,\, f \in N \tag{6}$$

The above definitions lead themselves to the following rectangular (shrunken) QAP version where n nodes are to be assigned to $m \leq n$ facility types, the u-th of which needs to allocate $\mu_u := |C_u|$ identical facility copies to different nodes:

$$\min\ \textstyle\sum_{i\in V}\sum_{u\in M}\sum_{j\in V}\sum_{v\in M}\tilde{a}_{uv}\, b_{ij}\, x_{iu}\, x_{jv} + \sum_{i\in V}\sum_{u\in M}\tilde{c}_{iu}x_{iu} \tag{7}$$

$$\textstyle\sum_{i\in V} x_{iu} = \mu_u \quad \forall u \in M \tag{8}$$
$$\textstyle\sum_{u\in M} x_{iu} = 1 \quad \forall i \in V \tag{9}$$
$$x_{iu} \in \{0,1\} \quad \forall i \in V,\, u \in M. \tag{10}$$

Note that variables x_{iu} are binary as we cannot assign more than one facility type to each node.

Theorem 1 *Model (7)-(10) is a valid reformulation of (1)-(4).*

Proof Our order of business is to prove that, for any feasible solution y of (1)-(4), one can define a feasible solution x of (7)-(10) of the same cost, and vice versa.

Because of constraints (3) and (9), the two solutions y and x will be described by specifying, for each $i \in V$, the assigned facility $f(i)$ in y and the facility type $u(i)$ in x.

Given y, solution x is obtained in a straightforward way by defining $u(i) := \theta(f(i))$ for all i. Similarly, given x we define y by choosing $f(i)$ inside $C_{u(i)}$ so as to never assign a same facility to two different nodes. This can be easily done, e.g., by initially marking as unused all facilities, and then scanning the nodes $i \in V$ in an arbitrary order. For each i, choose any unmarked facility $g \in C_{u(i)}$, mark it, and set $f(i) := g$. Because of (8), this procedure will always produce a feasible solution y for (1)-(4).

Note that while the solution x associated to y is unique, there are $\prod_{u\in M} \mu_u!$ different solutions y that can be associated with a given x. In any case, by construction, we have $\theta(f(i)) = u(i)$ for all $i \in V$.

Let $z(y)$ and $z(x)$ denote the cost of y and x in their respective models. As we are assuming $b_{ii} = 0$ for all i, and because of (5)-(6), we have

$$z(y) = \sum_{i\in V} \sum_{j\in V\setminus\{i\}} a_{f(i),f(j)}\, b_{ij} + \sum_{i\in V} c_{i,f(i)} = \tag{11}$$
$$\sum_{i\in V} \sum_{j\in V\setminus\{i\}} \tilde{a}_{\theta(f(i)),\theta(f(j))}\, b_{ij} + \sum_{i\in V} \tilde{c}_{i,\theta(f(i))} = \tag{12}$$
$$\sum_{i\in V} \sum_{j\in V\setminus\{i\}} \tilde{a}_{u(i),u(j)}\, b_{ij} + \sum_{i\in V} \tilde{c}_{i,u(i)} = z(x) \tag{13}$$

which concludes the proof. $\square$

Clone shrinking can be viewed a generalization of a method proposed in [Kaibel(1998)] for dealing with the case where $k \le n$ facilities have to be assigned to n nodes—as it happens, in particular, in the `esc` instances. In this case, the QAP instance contains $n - k$ *zero (or dummy) facilities* u such that $a_{uv} = a_{vu} = 0$ for all v. Kaibel's setting can therefore be viewed as a particular case of our framework arising when shrinking only the cluster induced by the zero facilities. Though beneficial, this latter operation is however not sufficient to remove the symmetry induced by other clone clusters (if any). For example, instance `tai64c` has 64 facilities that can be clustered in just two classes C_1 and C_2 with $|C_1| = 13$ and $|C_2| = 51$, and Kaibel's approach would shrink C_2 leaving the remaining 13 clones untouched.

It is worth observing that one could apply a similar reasoning to matrix b instead of a (as a matter of fact, this is equivalent to just apply a completely-general preprocessing step that swaps a and b while transposing c, an operation that leaves the optimization problem unchanged). However, we have verified that almost no clone exists with respect to the b matrices of the QAPLIB instances.

2.2. Symmetry in the distance matrix

Clones of course do not capture all possible sources of symmetry of a QAP instance, in particular because their definition disregards the distance matrix b completely. As a matter of fact, it is well known that the symmetry group of b for many QAP instances can lead to very wide orbits.

An extreme case arises when a single orbit of size n exists, meaning that one can always rename the nodes so as to map any node i to any other node j through a node permutation that leaves b unchanged. This "single b-orbit" property can easily be checked through standard software based e.g., on `nauty` [McKay(2010)], and we verified it holds true for all `esc` instances. An important implication is that one can always assume $x_{ifix,ufix} = 1$ for any arbitrary but fixed pair $(ifix, ufix) \in V \times M$. According to our computational experience, imposing such a fixing in a random way is likely to have a negligible impact in the computational effort to optimality, but choosing the pair in a careful way can be highly beneficial. In our study, we found that choosing $x_{ifix,ufix}$ as the one with the larger expected impact for branching (according to the criterion to be described in the next section) does in fact consistently lead to improved performance.

In any case, fixing a single variable $x_{ifix,ufix}$ at the root node does not remove all the symmetries arising from matrix b, so it is crucial to be able to detect and exploit the residual symmetries. As discussed in the next section, this can be done either by using a sound general-purpose solver that infers the symmetries from the input model, or by designing an ad-hoc solution method that derives the symmetries directly from the input data (in particular, from the distance matrix b).

3. Step Two: Mixed-integer linear models and algorithms

Clone shrinking is likely to be beneficial for any QAP solution method applied to instances involving clone facilities. Our approach was to generate and solve a suitable MILP model, by using both an off-the-shelf (black-box) MILP solver and an ad-hoc Branch&Cut scheme.

3.1. Choosing a suitable MILP formulation

Most MILP models for QAP work with additional 0-1 variables $\chi_{iujv} = y_{iu}y_{jv}$ that are used to linearize the quadratic objective function—the Adams-Johnson model [Adams and Johnson(1994)] being perhaps the best-known such formulation. These kinds of models require $\Theta(n^4)$ variables and $\Theta(n^3)$ constraints, so they become huge even for medium-size instances.

As our ultimate goal was to solve `esc128`, we decided to address more scalable models that do not become hopeless for large instances. In particular, we looked for MILP models requiring just $O(n^2)$ variables and constraints, and adapted them for our rectangular QAP version. An obvious candidate was the MILP model credited to Kaufman and Broeckx (KB) [Kaufman and Broeckx(1978)] that requires the introduction of just $n \cdot m$ additional (continuous) variables

$$w_{iu} = \Big(\sum_{j\in V}\sum_{v\in M} \tilde{a}_{uv} b_{ij} x_{jv}\Big)\ x_{iu} \tag{14}$$

that allow one to formulate the rectangular QAP as the nonlinear problem:

$$\min \quad \textstyle\sum_{i\in V}\sum_{u\in M} w_{iu} + \sum_{i\in V}\sum_{u\in M} \tilde{c}_{iu} x_{iu} \tag{15}$$

$$w_{iu} \ge \big(\textstyle\sum_{j\in V}\sum_{v\in M} \tilde{a}_{uv} b_{ij} x_{jv}\big) x_{iu} \quad \forall i \in V,\, u \in M \tag{16}$$

$$w_{iu} \ge 0\ \forall i \in V,\, u \in M \tag{17}$$

$$x \in X \tag{18}$$

where

$$X := \{x : (8)-(10) \ \text{ hold }\} \tag{19}$$

denotes the set of feasible solutions. Note that optimizing any linear function over X is just a rectangular LAP solvable in $O(n^3)$ time.

Nonlinear constraints (16) can be linearized as follows. For any given pair $(i,u) \in V \times M$, our order of business is to impose the disjunctive condition:

$$(x_{iu} = 1 \text{ and } w_{iu} \geq \sum_{j\in V}\sum_{v\in M} \tilde{a}_{uv} b_{ij} x_{jv}) \ \vee \ (x_{iu} = 0 \text{ and } w_{iu} \geq 0) \tag{20}$$

through linear inequalities that are valid for

$$P_{iu} := conv(P^0_{iu} \cup P^1_{iu})$$

where

$$P^1_{iu} := \{(x, w_{iu}) : x \in X,\ x_{iu} = 1,\ w_{iu} \geq \sum_{j\in V}\sum_{v\in M} \tilde{a}_{uv} b_{ij} x_{jv}\}$$

and

$$P^0_{iu} := \{(x, w_{iu}) : x \in X,\ x_{iu} = 0,\ w_{iu} \geq 0\}.$$

Note that, while one can easily optimize any linear function over P_{iu}, the polyhedral structure of P_{iu} is far from trivial and depends on the numerical entries of matrices $\tilde{a}$ and b. E.g., even for a toy case with $n = m = 5$ and $\tilde{a}_{uv} b_{ij} \in \{0, \cdots, 7\}$, we could verify through software PORTA [Christof(2009)] that P_{iu} has 1693 facets whose definition involves nasty coefficients (up to 84, in absolute value).

To get a compact formulation, in the basic linearized model one typically looks for a single constraint for each pair (i,u), that has the form

$$w_{iu} \geq \beta_0 + \sum_{j\in V}\sum_{v\in M} \beta_{jv} x_{jv} \tag{21}$$

and capable of imposing (20) whenever $x \in X$. In the KB model, this is achieved by just replacing the nonlinear constraints (16) by

$$w_{iu} \geq \sum_{j\in V}\sum_{v\in M} \tilde{a}_{uv} b_{ij} x_{jv} - SUM_{iu}(1 - x_{iu}) \quad \forall i \in V,\, u \in M \tag{22}$$

where $SUM_{iu} := \sum_{j\in V}\sum_{v\in M} \tilde{a}_{uv} b_{ij}$. This model is however known to be of little use in practice because it is heavily based on the big-M constraints (22). As a result, w and x variables have almost no link when x is allowed to be fractional, as it happens in the LP relaxation solved at each node of an enumerative solution method. As a matter of fact, it can be proved [Xia and Yuan(2006)] that the root-node bound is always zero with this model when $\tilde{c} = 0$ (as it happens in most cases).

A first improvement of the KP model has been recently proposed in [Xia and Yuan(2006)]. For the sake of completeness, we next rephrase the resulting model for our rectangular QAP version.

For a given pair $(i,u) \in V \times M$, let

$$INC(i,u) := \begin{cases} \{(j,v) : j = i \ \vee \ v = u\} \setminus \{(i,u)\} & \text{if } \mu_u = 1 \\ \{(j,v) : j = i\} \setminus \{(i,u)\} & \text{otherwise} \end{cases} \tag{23}$$

be the index set of those variables that are "incompatible" with x_{iu}, i.e., such that, for any $x \in X$, setting $x_{iu} = 1$ implies $x_{jv} = 0$ for all $(j,v) \in INC(i,u)$. It is immediate to see that $x_{iu} = 0$ implies $x_{jv} = 1$ for at least one $(j,v) \in INC(i,u)$.

Theorem 2 *([Xia and Yuan(2006)]) The following inequality*

$$w_{iu} \geq \sum_{j\in V}\sum_{v\in M} \beta_{jv}x_{jv} - MAX(\beta)(1-x_{iu})$$

is valid for P_{iu}, where for all $(j,v)\in V\times M$

$$\beta_{jv} := \begin{cases} 0 & \text{if } (j,v)\in INC(i,u) \\ \tilde{a}_{uv}b_{ij} & \text{otherwise} \end{cases} \tag{24}$$

and $MAX(\beta)$ is the maximum (rectangular) LAP value over β.

Proof The inequality would be obviously valid for P^1_{iu} without zeroing out coefficients β_{jv} for $(j,v)\in INC(i,u)$, that are however immaterial for validity with respect to P^1_{iu} because of the definition of $INC(i,u)$. Because of the correcting term $-MAX(\beta)(1-x_{iu})$, the inequality is also valid for P^0_{iu}, hence the claim. □

Note that Xia and Yuan compute $MAX(\beta)$ without imposing condition $x_{iu}=0$, although this would be mathematically correct and would lead to a possibly improved coefficient.

The Xia-Yuan model inherits from the KB one the "big-M trick" of imposing validity for P^0_{iu} by using a sufficiently large (in absolute value) negative coefficient for term $(1-x_{iu})$. However, equations (9) imply that this term is in fact equal to $\sum_{v\neq u} x_{iv}$, hence one can compute the coefficient of each x_{iv} $(v\neq u)$ in an independent way, thus obtaining a possibly improved inequality. More generally, for every (i,u) one can easily improve any given valid inequality of the form (21) by lifting, in any given sequence, the coefficients β_{jv} for $(j,v)\in INC(i,u)$. For all such (j,v), setting $x_{jv}=1$ implies $x_{iu}=0$, hence one needs to address validity w.r.t. P^0_{iu} only, and the lifting coefficient can easily be computed as

$$\beta_{jv} := \beta_{jv} - (\beta_0 + \max\{\beta x : x\in X, x_{jv}=1\}) \tag{25}$$

Indeed, the value of β_{jv} is immaterial for all $x\in X$ with $x_{jv}=0$. When $x_{jv}=1$, instead, validity w.r.t. P^0_{iu} requires $0\geq \beta_0+\max\{\beta x : x\in X, x_{jv}=1\}$, hence β_{jv} needs to be updated as in (25) so as to guarantee $0=\beta_0+\max\{\beta x : x\in X, x_{jv}=1\}$.

After a number of preliminary (sometimes quite disappointing) tests with different forms of (22), we arrived to the following conclusions:

- For each given pair (i,u), a single constraint of the form (22) remains quite weak no matter the way its coefficients are lifted—its intrinsic big-M nature forces this constraint to become relevant only when variable x_{iu} is forced to attain a value very close to 1 (e.g., because of branching), whereas any value not very close to 1 deactivates the constraint completely.
- The LP relaxation value at the root node is not a discriminating factor (all forms producing essentially the same LP bound), hence one should prefer a form that best fits the MILP solver, e.g., a sparse form with small coefficients.
- For highly-symmetric instances as the `esc` ones, it is extremely important to choose a formulation that does not hide the instance symmetry. In particular, over-sophisticated formulations (possibly involving clever classes of additional cuts) are perhaps better from a polyhedral point of view, but are prone to negative side effects such as destroying symmetry—thus making it impossible for a general-purpose MILP solver to recognize it and fully exploit it. The lesson learned is that, in some cases, the simpler the formulation, the more likely it will not introduce nasty side effects.

We therefore elaborated the Xia-Yuan model with the aim of producing a simple alternative with improved characteristics—in particular, coefficient sparsity. For each given (i,u), we start with the following reduced-cost version of (21)

$$w_{iu} \geq MX_{iu}\, x_{iu} + \sum_{j\in V}\sum_{v\in M} \overline{\beta}^{iu}_{jv} x_{jv} \tag{26}$$

where MX_{iu} is the value of the *maximum* LAP with $x_{iu}=1$ computed over costs $\tilde{a}_{uv}b_{ij}$, and $\overline{\beta}^{iu}_{jv}$ are the corresponding nonpositive LP reduced costs. To increase sparsity, we reset $\overline{\beta}^{iu}_{jv}=0$ for all $(j,v)\in INC(i,u)$, an operation that does not affect validity w.r.t. P^1_{iu}. We then consider all $(j,v)\in INC(i,u)$, in an arbitrary but fixed sequence, and make the inequality also valid for P^0_{iu} by recomputing $\overline{\beta}^{iu}_{jv}$ through lifting.

The above considerations are summarized in the following

Theorem 3 *For all $(i,u)\in V\times M$, inequality (26) is valid for P_{iu}, where MX_{iu} is the maximum LAP value over costs $\tilde{a}_{uv}b_{ij}$ when imposing $x_{iu}=1$, and $\overline{\beta}^{iu}$ is defined by first taking the (non-positive) reduced costs associated with MX_{iu}, then resetting $\overline{\beta}^{iu}_{jv}=0$ for all $(j,v)\in INC(i,u)$, and finally recomputing $\overline{\beta}^{iu}_{jv}$ for all $(j,v)\in INC(i,u)$, in any arbitrary but fixed sequence, by using the lifting formula*

$$\overline{\beta}^{iu}_{jv} := -\max\{\overline{\beta}^{iu}x : x\in X, x_{jv}=1\} \geq 0. \tag{27}$$

Proof The inequality is trivially valid for P^1_{iu} as the right-hand side values of (26) and (22) coincide for all $x\in X$ with $x_{iu}=1$ because of the basic property of LP reduced costs for equally-constrained LPs. As to validity w.r.t. P^0_{iu}, take any solution $x\in X$ with $x_{iu}=0$. As already observed, $x_{iu}=0$ implies $x_{jv}=1$ for at least one pair $(j,v)\in INC(i,u)$, so take the last such pair encountered in the lifting sequence. Because of (27), the right-hand side of (26) becomes nonpositive after lifting, i.e., the inequality is dominated by $w_{iu}\geq 0$ and then vanishes, as required. □

A main reason for the use of reduced $\overline{\beta}_{jv}$'s instead of the original β_{jv}'s is that they typically lead to sparser constraints with smaller coefficient values—both properties being beneficial for the solution of the model through a MILP solver.

A substantial improvement of the basic KB form, still due to Xia and Yuan [Xia and Yuan(2006)], is the introduction of the additional valid cuts

$$w_{iu} \geq MN_{iu}\ x_{iu},\ \forall i\in V,\, u\in M \tag{28}$$

where MN_{iu} denotes the *minimum* rectangular LAP value over costs $\tilde{a}_{uv}b_{ij}$ when fixing $x_{iu}=1$. Note that these cuts do not suffer from the big-M drawback—a small nonzero fractional value for x_{iu} is not able to completely deactivate them. Indeed, the LP relaxation of the model involving these cuts gives, at least, the so-called Gilmore-Lawler [Gilmore(1962), Lawler(1963)] bound for QAP—while the original KB model fails to give such bound by a large amount.

A last improvement is proposed in [Zhang et al.(2010)Zhang, Beltran-Royo, and Ma], where the Xia-Yuan model is refined to deal implicitly with cuts (28) by working with the shifted variables

$$\sigma_{iu} := w_{iu} - MN_{iu}\ x_{iu} \geq 0. \tag{29}$$

After a number of tests, we found that the following new MILP formulation gives the best performance in our setting.

$$\min \textstyle\sum_{i\in V}\sum_{u\in M}\sigma_{iu}+\sum_{i\in V}\sum_{u\in M}(\tilde{c}_{iu}+MN_{iu})x_{iu} \tag{30}$$

$$\sigma_{iu}\geq (MX_{iu}-MN_{iu})x_{iu}+\textstyle\sum_{j\in V}\sum_{v\in M}\overline{\beta}^{iu}_{jv}x_{jv}\ \ \forall i\in V,\, u\in M \tag{31}$$

$$\sigma_{iu}\geq 0,\quad \forall i\in V,\, u\in M \tag{32}$$

$$x\in X \tag{33}$$

where, for every (i,u), coefficients MX_{iu}'s and $\overline{\beta}^{iu}_{jv}$'s are defined as in Theorem 3.

As to the variable $x_{ifix,ufix}$ possibly fixed to one because of symmetry (see Section 2.2), if any, we include this fixing in the definition of X so as to tighten the coefficients of all constraints (31).

Finally, it is easy to verify that the optimal QAP solution value must be an even integer in case matrices a,b,c are integer, a and b are symmetric, and c itself is even. In this case, one can slightly improve formulation (30)-(33) by rewriting the objective function as just $2z$ and require

$$2z=\sum_{i\in V}\sum_{u\in M}\sigma_{iu}+\sum_{i\in V}\sum_{u\in M}(\tilde{c}_{iu}+MN_{iu})x_{iu},\quad z\text{ integer.}$$

3.2. Use of a general-purpose MILP solver as a black box

Our first quick shot was to write our MILP model (30)-(33) in a file for each `esc` instance, and to solve it through a black-box commercial MILP solver—`IBM ILOG Cplex 12.2` in our case. We also included the previously-unsolved instance `tai64c` in our test-bed because this instance is particularly suited for our clone-shrinking approach—the number of its facilities being reduced from $n=64$ to just $m=2$.

We provided the black-box solver with a branching-priority input file where the branching priority of variable x_{iu} is a measure of the impact of x_{iu} into our model—and specifically on constraints (31). This is very much in the spirit of the recent work by Chinneck and co-authors [Patel and Chinneck(2007), Pryor and Chinneck(2011)]. As each constraint (31) has a dominating coefficient $MX_{iu}-MN_{iu}$ whose absolute value is typically (integer and) much larger than that of all other coefficients, we used the score

$$10^6(MX_{iu}-MN_{iu})+10^3u+i$$

as our branching priority for x_{iu}. Note that variables x_{iu} with $MX_{iu}-MN_{iu}=0$, if any, receive the lowest score, which is very reasonable in that constraint (31) becomes redundant as it requires $\sigma_{iu}\geq 0$ even for $x_{iu}=1$, hence $\sigma_{iu}=0$ in any optimal solution. This is true, in particular, when u is a zero facility, i.e., $\tilde{a}_{uv}=\tilde{a}_{vu}=0$ for all v.

According to our computational experience, the use of the above priorities has a strong impact on the MILP solver performance, allowing for the solution of instances that are out of reach for the default—even strong—branching strategies. We are confident that this new branching criterion will prove quite useful within other solution approaches as well.

Table 1 reports results obtained by using `IBM ILOG Cplex 12.2` for solving some easy and hard `esc` instances (plus `tai64c`) to proven optimality. These experiments were performed on a quad-core Intel Xeon CPU running at 3.2 GHz under Mac OS 10.6 and equipped with 16GB of RAM. According to our preliminary tests, Cplex typically finds the optimal solutions within negligible computing time for these instances, whereas cuts seem to be counterproductive. Therefore in our runs Cplex's cuts and heuristics were disabled, and no upper cutoff was specified on input (we actually used a cutoff of 10^8 just to help preprocessing to cleanup our model). Instead, the use of the most aggressive setting for symmetric reductions (level 5) proved to be highly beneficial.

For all cases, the optimal solution turned out to be of the same value as the best-known ones reported in QAPLIB. All the input (.lp and .ord) files as well as the log's of these runs are available, on request, from the authors.

Table 1 Optimal solutions found by `IBM ILOG Cplex 12.2` in interactive mode (8 threads on a quad-core Intel Xeon CPU@3.2GHz with 16GB RAM running under MacOS 10.6). Instances marked by $*$ are reported as unsolved in QAPLIB.

Instance	n	m	OPT	time (s)	#nodes
`esc16a`	16	9	68	0.35	4,133
`esc16b`	16	7	292	3.07	71,075
`esc16c`	16	12	160	130.98	2,652,014
`esc16d`	16	12	16	0.51	10,796
`esc16e`	16	8	28	0.05	421
`esc16f`	16	1	0	0.00	0
`esc16g`	16	9	26	0.04	450
`esc16h`	16	5	996	0.23	4,967
`esc16i`	16	10	14	0.18	3,216
`esc16j`	16	7	8	0.03	114
`esc32c`*	32	10	642	9,643.82	81,650,962
`esc32d`*	32	13	200	2,973.26	12,757,770
`esc32e`	32	6	2	0.04	70
`esc32g`	32	7	6	0.06	597
`esc64a`*	64	15	116	509.87	1,206,370
`tai64c`*	64	2	1,855,928	18,250.40	1,216,074,081

3.3. Designing an ad-hoc Branch&Cut algorithm

Our next step was the design of an ad-hoc Branch&Cut code, aimed at achieving a better control on additional cuts, branching strategy, and symmetry breaking.

Local cuts

As far as cuts are concerned, at some branching nodes we generated additional local cuts of the form

$$\sigma_{iu} \geq (MN_{iu}^{+} - MN_{iu})\, x_{iu}$$

where MN_{iu}^{+} is defined as MN_{iu}, but takes the current variable fixings into account. In our implementation, only variables fixed to 1 were considered, hence a very fast $O(n)$ algorithm using pre-sorted arrays was used to compute MN_{iu}^{+}—very much in the spirit of the classical Gilmore-Lawler bound computation [Burkard et al.(2009)Burkard, Dell'Amico, and Martello]. As the overhead introduced by the addition of new cuts to the LP (rather than the separation time itself) is quite substantial, we only performed cut generation at the nodes where the number of variables fixed to 1 was even and in the range [2..16], and we added them to the node relaxation only if the number of violated cuts found was not smaller than 5.

Branching strategy

As to branching, we used a standard 2-way branching scheme on a fractional variable $x_{ib,ub}$ (say) selected as follows. The input priority file described the previous section was used to select facility ub. As to the choice of ib, we selected the fractional variable that minimizes the quantity

$$\sum_{(j,v)\in F} \tilde{a}_{ub,v} b_{ij}$$

where F is the set of pairs (j,v) for which x_{jv} is fixed to 1 at the current branching node. The rationale behind this choice is that such a variable gives the smallest contribution to the overall cost with respect to the fixed variables, and as such it is likely to take value 1 on any "good" solution in the current subtree. For this reason, fixing it to zero is likely to increase the bound significantly, thus strengthening the weak zero-branch.

Dealing with residual symmetries

Given the choice of the branching variable, we implemented a specialized orbital branching and fixing scheme [Ostrowski et al.(2011)Ostrowski, Linderoth, Rossi, and Smriglio], computing the symmetry groups and orbits associated with the distance matrix b. To ensure correctness of the resulting code, all Cplex's symmetry features and dual reductions were deactivated by setting `CPX_PARAM_SYMMETRY` to 0 and `CPX_PARAM_REDUCE` to `CPX_PREREDUCE_PRIMALONLY`.

The basic idea of our specialized orbital branching implementation is as follows. Given a node γ of the enumeration tree, let $F_1(\gamma) \subset V$ be the set of nodes that are fixed to some facility (e.g., because of branching). We compute the *pointwise-stabilizer* $stab(F_1(\gamma))$ as the subgroup of the symmetry group of b that keeps each element in $F_1(\gamma)$ fixed. Given a branching variable x_{iu}, we then compute the orbit O_i with respect to $stab(F_1(\gamma))$, i.e., the set of nodes that are equivalent to i according to $stab(F_1(\gamma))$. Finally, we enforce the branching dichotomy:

$$(x_{iu} = 1) \vee (x_{ju} = 0 \ \forall j \in O_i)$$

Stabilizers $stab(F_1(\gamma))$ and orbits O_i could be computed by general purpose tools such as `nauty` [McKay(2010)] or `saucy` [Darga et al.(2008)Darga, Katebi, Liffiton, Markov, and Sakallah]. However, as long as `esc` instances are addressed, one can exploit the very special structure of matrix b. Indeed, b_{ij} is defined as the Hamming distance $\Delta(i,j)$ between the binary representations of the two k-bit numbers $i-1$ and $j-1$, minus 1. As such, any bijective function ϕ from the set K of k-bit numbers to itself (a permutation) that preserves the Hamming distance, i.e., such that

$$\Delta(\phi(i),\phi(j)) = \Delta(i,j) \quad \forall i,j \in K,$$

can be applied to the rows and columns of b, leaving matrix b unchanged. As a matter of fact, it is known [Anstreicher(2003)] that the following basic operations (and their compositions) preserve the Hamming distance Δ. Given a k-bit string $d_{k-1}\cdots d_1 d_0$:

- (*flip*) flip a given subset T of its bits; e.g., given the 4-bit string $d_3d_2d_1d_0 = 1001$ and the subset $T = \{0,2\}$ we obtain $d_3\overline{d}_2 d_1 \overline{d}_0 = 1100$;
- (*permute*) apply a given bit permutation π; e.g., given the 4-bit string $d_3d_2d_1d_0$ and the bit permutation (in cycle notation) $\pi = (0,1,2)(3)$, we obtain $d_3d_1d_0d_2$.

Because of the above symmetries, when selecting the variable $x_{ifix,ufix}$ to be fixed to 1, one can always assume $ifix = n$, whose associated bit-string is $11\cdots 1$. This particular choice removes all bit flips from the set of feasible operations for computing the stabilizers, so we are left with permutations only. Note that we are considering permutations of very short bit strings, and the number of bit permutations is quite small—for our largest instance, `esc128`, $n = 128$ requires 7 bits, so there are just $7! = 5{,}040$ possible permutations. We can therefore afford to explicitly pre-compute and store all possible permutations to be used, at a later time, to compute stabilizers and orbits at each branching node. To be more specific, at each branching node we start with the pre-computed list of permutations and filter those that keep each node of $F_1(\gamma)$ fixed. We then compute the orbit O_i of a given node i by just applying the filtered permutations to the binary representation of i.

Results

Table 2 reports results when using our B&C code on the hardest `esc` instances of Table 1, on the same hardware. As expected, our Branch&Cut code outperforms the black-box `IBM ILOG Cplex 12.2` solver by a large amount, and it is able to solve `esc32c`, `esc32d` and `esc64a` in less than half an hour in total, on a single quad-core PC. On these instances, the speedup in running time is about 8×, while the reduction in the number of nodes is approximately 24×.

Table 2 Optimal solutions found by our specialized Branch&Cut code built on top of `IBM ILOG Cplex 12.2` (8 threads on a quad-core Intel Xeon CPU@3.2GHz with 16GB RAM running under MacOS 10.6). All these instances are reported as unsolved in QAPLIB.

Instance	n	OPT	time (s)	#nodes
`esc32c`	32	642	1156.00	3,102,322
`esc32d`	32	200	472.68	685,159
`esc64a`	64	116	83.52	143,124

4. Step Three: Exploiting a facility-flow splitting scheme

During our various attempts to solve `esc128` we observed the following apparently strange behavior.

The best-known upper bound for `esc128` is 64, while the Gilmore-Lawler bound is just 2. The root-node bound for our MILP model is 2 or 4, depending on the choice of the variable that can safely be fixed to 1 because of symmetry. Depending on the branching order, our enumerative method was able to improve this bound quite quickly, reaching lower bound values of about 40 after a few minutes of computation if a sound branching order was chosen.[1]

However, if we deliberately set to zero some nonzero entries of the flow matrix a (chosen in a proper way), a bound of 48 could be proved in about 2 seconds, after having enumerated just 2,000 nodes. In other words, we had the evidence that *reducing* the costs can in fact *improve* the bound obtained after a fixed amount of enumeration. This is counterintuitive, in that for any given branching node the lower bound computed w.r.t. the reduced flow matrix cannot be better than that computed w.r.t. the original one, and we do not expect that having a worse bound can help enumeration.

The explanation is that the modified matrix a induced symmetries that were not present in the original model, hence a large number of nodes could implicitly be pruned because of symmetry (as opposed to bound) considerations. This is particularly true after the application of our clone-shrinking mechanism: two "almost indistinguishable" facilities that could not be shrunk in the original problem, became clones in the reduced one and hence could be collapsed—thus saving a huge amount of enumeration.

The above considerations prompted us to analyze a facility-flow splitting scheme that is based on the following theorem. To ease notation, assume $c = 0$. We denote by QAP(α) the QAP instance (before clone shrinking) defined by flow matrix α, and by $opt(\alpha)$ its optimal solution value.

[1] As an exercise, we tried 200 random perturbation of our branching priority order: the bound available after 50,000 enumeration nodes using `IBM ILOG Cplex 12.2` in interactive mode ranged from just 4 to 46.

Theorem 4 *Let a^1 and a^2 be nonnegative matrices such that $a = a^1 + a^2$. Then $opt(a^1) + opt(a^2) \leq opt(a)$.*

Proof Let y any feasible solution to model (1)-(4). Then

$$\sum_{i \in V} \sum_{u \in M} \sum_{j \in V} \sum_{v \in M} a_{uv}\, b_{ij}\, y_{iu}\, y_{jv} =$$

$$\sum_{i \in V} \sum_{u \in M} \sum_{j \in V} \sum_{v \in M} a^1_{uv}\, b_{ij}\, y_{iu}\, y_{jv} + \sum_{i \in V} \sum_{u \in M} \sum_{j \in V} \sum_{v \in M} a^2_{uv}\, b_{ij}\, y_{iu}\, y_{jv}$$

i.e., the cost of y in QAP(a) is equal to the cost of y in QAP(a^1) plus its cost in QAP(a^2). The inequality sign in $opt(a^1) + opt(a^2) \leq opt(a)$ comes from the fact that we do not impose the optimal solution of QAP(a) to be coincident with the optimal solutions of QAP(a^1) and QAP(a^2)—imposing this condition would lead to an exact variable-splitting reformulation. $\square$

Theorem 4 remains obviously valid for nonnegative flow matrices a^1 and a^2 such that $a \geq a^1 + a^2$. In addition, the decomposition scheme can be iterated any number of times.

At first glance, computing a lower bound on the optimal QAP value by solving two QAP instances of the same size does not seem a clever idea. However, as already observed, for an effective solution of a MILP instance through an enumerative method, the *structure* of the instance matters even more than its size, so the real question is: Can we find a clever way to define two nonnegative flow matrices a^1 and $a^2 := a - a^1$ so as to obtain a structure that simplifies the exact solution process of both QAP(a^1) and QAP(a^2)? We tested two possible answers to the above question, both aimed at increasing the number of clones in both a^1 and a^2 (we assume a integer, as it is the case for all QAPLIB instances):

(i) define a^1 as the support of a, i.e., $a^1_{fg} := \min\{a_{fg}, 1\}$

(ii) select a facility subset $S \subset N$ and define $a^1_{fg} := 0$ if $f \in S$ or $g \in S$, $a^1_{fg} := a_{fg}$ otherwise.

The rationale of criterion (i) is clear: if we only distinguish between zero and nonzero entries in a^1 we increase the occurrence of clones, whereas a^2 becomes sparser than a and hence is likely to contain more clones.

Analogously, by using criterion (ii) all the facilities in S become zero-facilities w.r.t. a^1 and hence will be shrunk, whereas a^2 will be significantly sparser than a (and thus likely to contain more clones). In order to have a tight bound, a clever choice is to select S as a facility subset that contains no zero-facilities w.r.t. a, and such that the total flow passing through the cut $(S, N \setminus S)$ is as close to zero as possible.

By applying splitting criterion (i) to instance `esc32h` we were able to solve both QAP(a^1) and QAP(a^2) quite efficiently (on our PC, the B&C code took 4 and 7,795 seconds for the enumeration of 39,183 and 37,114,507 nodes, respectively), obtaining a lower bound of $opt(a^1) + opt(a^2) = 340 + 98 = 438$ that is equal to the value of the best-know feasible solution reported in [Burkard et al.(1991)Burkard, Karisch, and Rendl], thus proving its optimality.

By applying splitting criterion (ii) with

$$S := \{5, 28, 47, 48, 51, 53, 81, 87, 88, 94, 111, 113\}$$

to instance `esc128` (see Figure 2) we were able to solve both QAP(a^1) and QAP(a^2) very efficiently (our B&C code took just 2 and 7 seconds for the enumeration of 643 and 8,981 nodes, respectively), obtaining a bound of $opt(a^1) + opt(a^2) = 16 + 48 = 64$ that is equal to the value of the best-know feasible solution reported in [Burkard et al.(1991)Burkard, Karisch, and Rendl] (a feasible solution of value 64 can also be found in matter of seconds by our code), thus proving its optimality.

As to `esc32a`, by using criterion (ii) with

$$S := \{8, 12, 16, 17, 19, 21, 23, 24, 25, 26, 27\}$$

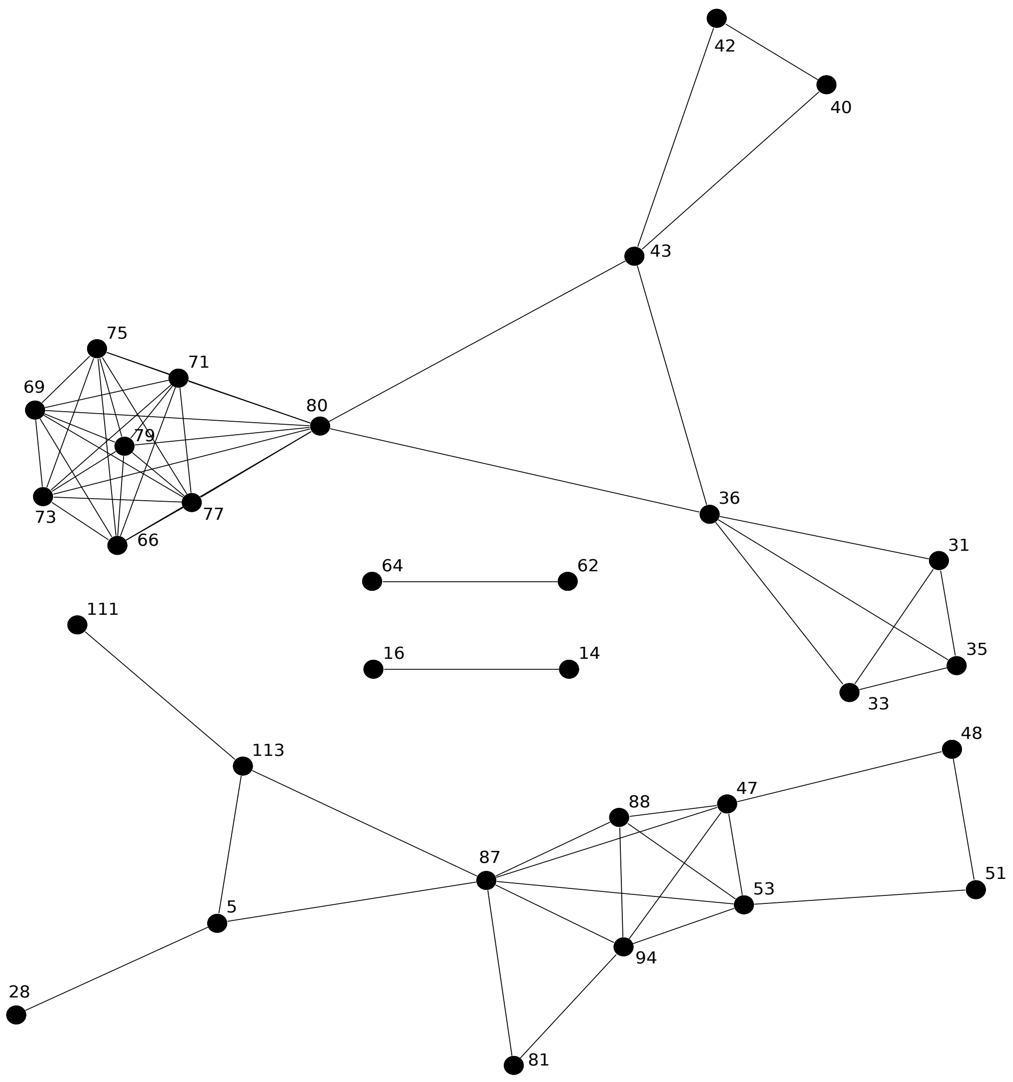

Figure 2 Support of the flow matrix a for instance `esc128` (zero facilities not shown).

(see Figure 3) we could again solve both QAP(a^1) and QAP(a^2) very efficiently (6 and 45 seconds for the enumeration of 19,247 and 201,178 nodes, respectively), obtaining a very tight bound of $opt(a^1) + opt(a^2) = 68 + 60 = 128$. As the best-known solution reported in [Burkard et al.(1991)Burkard, Karisch, and Rendl] has a value of 130, the (even) optimal solution value must be equal to either 128 or 130.

Table 3 summarizes the results obtained through our facility-flow splitting scheme. Computing times (in seconds) and numbers of nodes give the sum over the two runs of our Branch&Cut code to solve QAP(a^1) and QAP(a^2). Computing times do not include the time needed to generate and write the input MILP model files—this task required negligible time for `esc32*`, and about one minute for `esc128`.

5. Conclusions and future work

Three ideas toward the solution of hard QAP instances have been presented and evaluated experimentally on hard cases. To our pleasant surprise, at least for `esc` instances these ideas produced very good computational results even when embedded in a very basic solution scheme using off-the-shelf MILP technology.

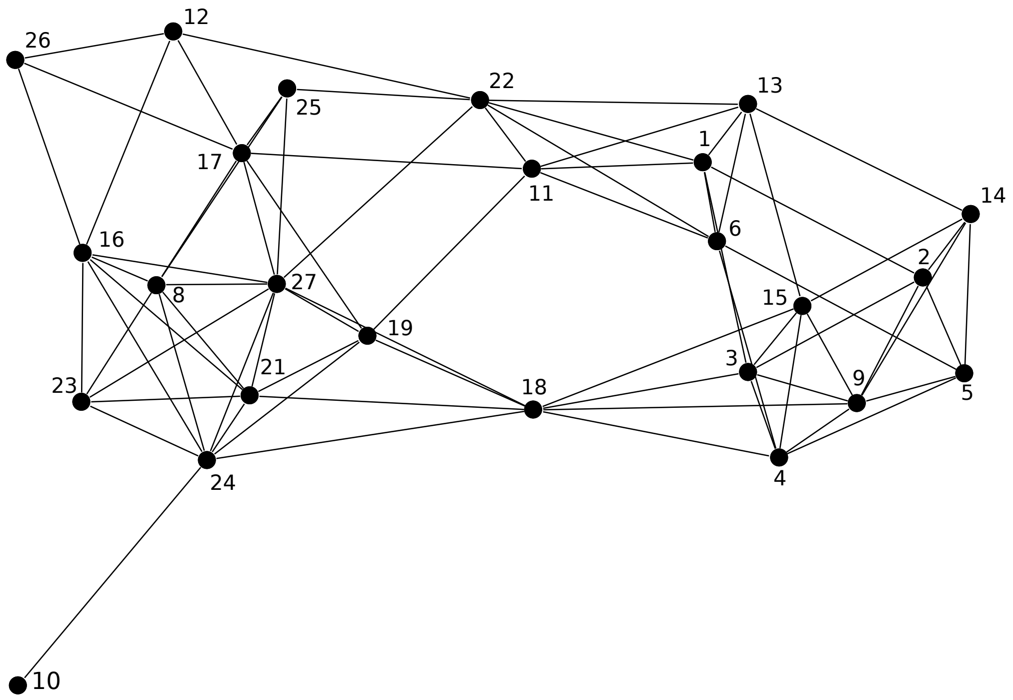

Figure 3 Support of the flow matrix a for instance `esc32a` (zero facilities not shown).

Table 3 Results obtained by our specialized Branch&Cut code through facility-flow splitting (8 threads on a quad-core Intel Xeon CPU@3.2GHz with 16GB RAM running under MacOS 10.6). All these instances are reported as unsolved in QAPLIB.

Instance	n	UB	LB	time (s)	#nodes
`esc32a`	32	130	60+68=128	51.16	220,425
`esc32h`	32	438	340+96=438	7,799.90	37,153,690
`esc128`	128	64	48+16= 64	9.63	9,624

We were able to solve to proven optimality all `esc` instances, with the only exception of `esc32a` (for which we proved a bound of 128 out of 130) and `esc32b`. In particular we solved, for the first time and in a matter of seconds, the “big fish” `esc128`—this is the largest QAPLIB instance ever solved to proven optimality [Burkard et al.(1991)Burkard, Karisch, and Rendl].

Future work should address the solution of the still-unsolved `esc` instances, and also the applicability of our ideas to other classes of still-unsolved QAP instances.

Acknowledgments

Research supported by the *Progetto di Ateneo* on “Computational Integer Programming” of the University of Padova. We thank Gianfranco Bilardi and Enoch Peserico for the use of the computers of the Center of Excellence “Scientific and Engineering Applications of Advanced Computing Paradigms”.

References

[Adams and Johnson(1994)] Adams, W.P., T.A. Johnson. 1994. Improved linear programming-based lower bounds for the quadratic assignment problem. *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 16. American Mathematical Society, 43–75.

[Anstreicher(2003)] Anstreicher, K. M. 2003. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming* **97** 27–42.

[Anstreicher et al.(2002)Anstreicher, Brixius, Goux, and Linderoth] Anstreicher, K.M., N.W. Brixius, J.-P. Goux, J. Linderoth. 2002. Solving large quadratic assignment problems on computational grids. *Mathematical Programming* **91** 563–588.

[Balas and Fischetti(1993)] Balas, E., M. Fischetti. 1993. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Mathematical Programming* **58** 325–352.

[Burkard et al.(2009)Burkard, Dell'Amico, and Martello] Burkard, R.E., M. Dell'Amico, S. Martello. 2009. *Assignment Problems*. SIAM. URL `http://www.ec-securehost.com/SIAM/OT106.html`.

[Burkard et al.(1991)Burkard, Karisch, and Rendl] Burkard, R.E., S. Karisch, F. Rendl. 1991. QAPLIB – A quadratic assignment problem library. *European Journal of Operational Research* **55** 115–119. URL `http://www.seas.upenn.edu/qaplib/`.

[Cela(1998)] Cela, E. 1998. *The quadratic assignment problem: Theory and algorithms*. Kluwer Academic Publishers, Dordrecht, The Netherlands. URL `http://www.ams.org/mathscinet-getitem?mr=99a%3a90001`.

[Christof(2009)] Christof, T. 2009. Porta. URL `http://typo.zib.de/opt-long_projects/Software/Porta/`.

[Clausen and Perregaard(1997)] Clausen, J., M. Perregaard. 1997. Solving large quadratic assignment problems in parallel. *Computational Optimization and Applications* **8** 111–127.

[Darga et al.(2008)Darga, Katebi, Liffiton, Markov, and Sakallah] Darga, P. T., H. Katebi, M. Liffiton, I. Markov, K. Sakallah. 2008. Saucy. URL `http://vlsicad.eecs.umich.edu/BK/SAUCY/`.

[de Klerk and Sotirov(2010)] de Klerk, E., R. Sotirov. 2010. Exploiting group symmetry in semidefinite programming relaxations of the quadratic assignment problem. *Mathematical Programming* **122** 225–246.

[Eschermann and Wunderlich(1990)] Eschermann, B., H. J. Wunderlich. 1990. Optimized synthesis of self-testable finite state machines. *20th International Symposium on Fault-Tolerant Computing (FFTCS 20)*. 390–397.

[Fischetti et al.(2011)Fischetti, Monaci, and Salvagnin] Fischetti, M., M. Monaci, D. Salvagnin. 2011. Three ideas for the quadratic assignment problem. *CPAIOR 2011, Late Breaking Abstracts*. ZIB report 11-20, 9–13. URL `http://cpaior2011.zib.de/downloads/CPAIOR2011_abstracts.pdf`.

[Gilmore(1962)] Gilmore, P.C. 1962. Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics* **14** 305–313.

[Kaibel(1998)] Kaibel, V. 1998. Polyhedral combinatorics of quadratic assignment problems with less objects than locations. R. Bixby, E. Boyd, R. Ríos-Mercado, eds., *Integer Programming and Combinatorial Optimization*, *Lecture Notes in Computer Science*, vol. 1412. Springer Berlin / Heidelberg, 409–422.

[Kaufman and Broeckx(1978)] Kaufman, L., F. Broeckx. 1978. An algorithm for the quadratic assignment problem using Benders' decomposition. *European Journal of Operational Research* **2** 204–211.

[Koopmans and Beckmann(1957)] Koopmans, T.C., M.J. Beckmann. 1957. Assignment problems and the location of economic activities. *Econometrica* **25** 53–76.

[Lawler(1963)] Lawler, E.L. 1963. The quadratic assignment problem. *Management Science* **9** 586–599.

[Margot(2002)] Margot, F. 2002. Pruning by isomorphism in branch-and-cut. *Mathematical Programming* **94** 71–90.

[McKay(2010)] McKay, B.D. 2010. Nauty users guide (version 2.4). Technical report, Dept. Comp. Sci., Australian National University.

[Ostrowski et al.(2011)Ostrowski, Linderoth, Rossi, and Smriglio] Ostrowski, J., J. Linderoth, F. Rossi, S. Smriglio. 2011. Orbital branching. *Mathematical Programming* **126** 147–178.

[Patel and Chinneck(2007)] Patel, J., J.W. Chinneck. 2007. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming* **3** 445–474.

[Pryor and Chinneck(2011)] Pryor, J., J.W. Chinneck. 2011. Faster integer-feasibility in mixed-integer linear programs by branching to force change. *Computers & OR* **38** 1143–1152.

[Xia and Yuan(2006)] Xia, Y., Y.X. Yuan. 2006. A new linearization method for quadratic assignment problem. *Optimization Methods and Software* **21** 803–816.

[Zhang et al.(2010)Zhang, Beltran-Royo, and Ma] Zhang, H., C. Beltran-Royo, L. Ma. 2010. Solving the quadratic assignment problem by means of general purpose mixed integer linear programming. URL `http://www.optimization-online.org/DB_HTML/2010/05/2622.html`.