Quadratic Semi-Assignment Problems on Structured Graphs

by

Federico Malucelli* and Daniele Pretolani*

Abstract: *The paper presents a class of polynomially solvable instances of Quadratic Semi-Assignment Problem. Some properties of this class are illustrated. In the light of these results lower bounds and solution techniques for the general case are proposed. A computational comparison of some lower bounds is reported.*

Keywords: *Quadratic Semi-Assignment Problem, Graph Theory, Lower Bounds, Branch and Bound.*

1. Introduction

In this paper we will study some new solution approaches to the Quadratic Semi-Assignment Problem (QSAP). In particular we will discuss a class of problems that can be solved in polynomial time and how these results can be exploited to provide lower bounds and solution algorithms for the general case.

In order to illustrate the problem we will use the following application example: consider a distributed computing system with p not necessarily identical processors, and n processes to be assigned to the processors. Let us call N the set of processes and M the set of processors. The following data are known:

- during the computation processes i and j exchange f_{ij} units of information;
- the time needed to move one unit of information from processor s to processor r is d_{rs};
- the computation time required by process i when it runs on processor s is e_{is}.

The *mapping problem* is that of assigning the processes to the processors so that the global time spent by the system (execution and communication time) is minimized. Let Π be the set of all the feasible assignment functions $\rho: N \rightarrow M$ which associate a processor $\rho(i) \in M$ to each process $i \in N$; the problem can be formulated as a QSAP as follows:

$$Z = \min \{ \sum_{i,j \in N} f_{ij} d_{\rho(i)\rho(j)} + \sum_{j \in N} e_{i\rho(i)},\ \rho \in \Pi \}. \qquad (1.1)$$

The non zero f_{ij} coefficients define the communication pattern between processes, usually represented by an undirected graph. We will show that the optimal mapping

* Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 PISA, Italy

can be found in polynomial time when this graph belongs to the class of *reducible graphs*.

The QSAP can be also formulated in a more general way: considering the matrix q_{ijhk} $i,j \in N$, $h,k \in M$, the problem is:

$$Z = \min \{ \sum_{i,j \in N} q_{ij\rho(i)\rho(j)}, \rho \in \Pi \}. \qquad (1.2)$$

Throughout the paper, we will consider the first formulation of QSAP; however, our results can be easily extended to the more general formulation.

Many other problems can be formulated as QSAP, for example clustering and partitioning [10], assignment of professors to departments [8], some scheduling problems [5].

The QSAP is well known to be NP-hard [13]; some lower bounds for the problem have been devised in [8] and [9]. Polynomial classes are presented in [3], [4] studying distributed computing systems. Sometimes the model has been complicated to take into account "real life" factors as in [6], [15].

The paper is organized as follows. In section 2 we will introduce the class of reducible graphs and we will present some of their fundamental properties as well as recognition algorithms. In section 3 we will devise an algorithm for solving instances of QSAP whose associated graph is reducible. In section 4 we will propose some ideas for exploiting the results of section 3 in solving general cases of QSAP, in particular we will focus on lower bounds. Finally in section 5, some preliminary computational results will be reported.

In the paper we will use the standard graph terminology introduced in [2].

2. The class of reducible graphs

Consider the undirected graph $G(N,A)$ where $n=|N|$ and $m=|A|$; G is *reducible* if and only if it can be reduced to a single node by the following operations:

- *Tail reduction*

  let i be a node of degree 1 (i.e. there is only one arc incident with node i) and (j,i) be the arc connecting node i to the rest of the graph G. The graph G can be reduced to a new graph G' where node i and arc (j,i) have been deleted (see fig. 1). We will denote with Tail(i) the above reduction operation.

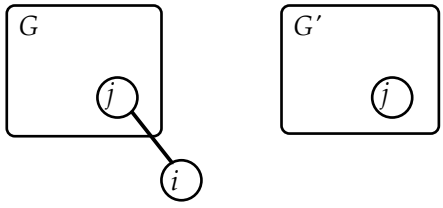

fig. 1

- *Series reduction*
  let i be a node of degree 2 and let (i,j) and (i,h), $j \neq h$, the two arcs incident with i; the graph G can be reduced to a new graph G' obtained from G by removing node i, arc(j,i), arc (i,h) and adding a new arc (j,h) (see fig. 2). This operation is denoted by Series(h, i, j).

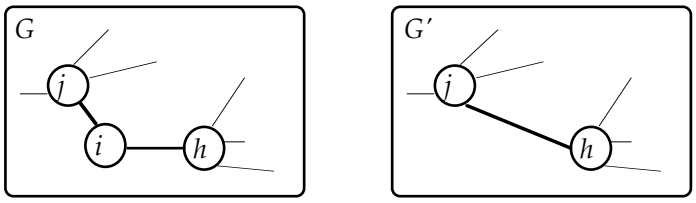

fig. 2

- *Parallel reduction*
  let $a=(i,j)$ and $a'=(i,j)$ be two "parallel" arcs of graph G. The graph can be reduced to a new graph G' with a single arc between nodes i and j (see fig. 3). This operation is denoted by Parallel(i, j).

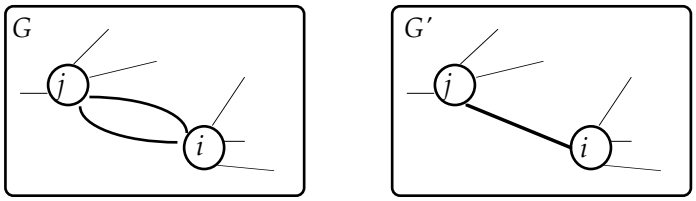

fig. 3

It is easy to see that the class of reducible graphs is a proper extension of the class of *series-parallel* graphs, which are reducible to a single arc by a sequence of parallel and series reductions ([16]). For example, trees are included in the class of reducible graphs, although they do not belong to the series-parallel class.
In the following we introduce some properties of reducible graphs. These properties are known to hold for series-parallel graphs.

Property 1
Given a graph G, assume that two different reduction operations, l_1 and l_2, can be applied to G, respectively obtaining the graphs G_1 and G_2. Then there exist two reductions l_1' and l_2' that can be applied to G_1 and G_2, respectively, giving two graphs G_1' and G_2' which are isomorphic.
Proof
The property is obvious if l_1 and l_2 apply to disjoint sets of nodes; in this case, $l_1' = l_2$ and $l_2' = l_1$. The same is true if either l_1 or l_2 is a parallel reduction; the only interesting cases arise when series and/or tail reductions are applicable to the same

set of nodes. For example, let l_1 = Series(i, j, k) and l_2= Tail(k); l_1 is no more applicable to G_2, but we can set l_1' = Tail(k) and l_2' = Tail(j) (see fig 4). Note that G_1 and G_2 are equal if node k in G_1 is identified with node j in G_2. A similar example can be given when l_1 = Series(i, j, k) and l_2= Series(h, i, j), and when two different tail reductions are applied. □

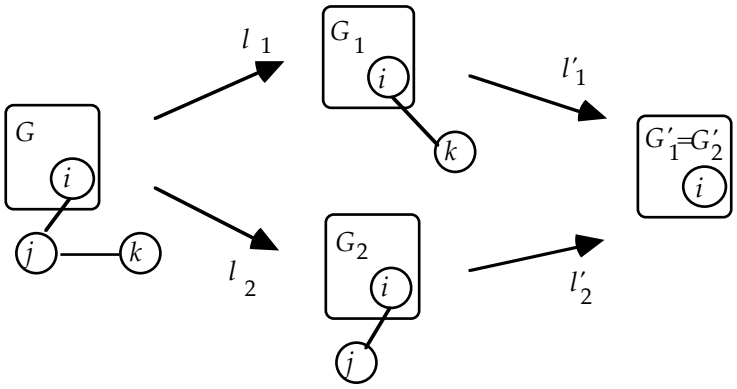

fig. 4

Property 2

Suppose repeatedly applying reductions to a graph G as far as possible; the graph G' obtained is independent of the sequence of reductions.

Proof

Let G_1 and G_2 be two different graphs obtained from G applying two different sequences of reductions. By repeatedly applying Theorem 1, we can reduce G_1 and G_2 to a pair of graphs G_1' and G_2' which are isomorphic. Hence, if no reduction operations can be applied to G_1 and G_2, it follows that G_1 and G_2 must be isomorphic. □

Property 2 means that if different reductions are applicable to the same graph, we can break ties arbitrarily without affecting the final result. This establishes the *confluence* property of the reducing operations; this property is well known for series-parallel graphs [7].

Property 3

If G is a connected graph, any graph G', obtained from G by reduction, is connected.

Property 4

Every induced subgraph G' of a reducible graph G is reducible.

Proof

Since G is reducible, there is a sequence $\{l_1,\ldots,l_k\}$ of reductions that transforms G into a single node. A sequence $\{l_1',\ldots,l_k'\}$ reducing G' to a single node can be obtained as follows. If the nodes and the arcs of G involved by the operation l_i are present in G' then $l_i'=l_i$. Otherwise the following cases must be distinguished:

- if l_i is a Parallel or Tail reduction, then l_i' is the "empty" reduction, i.e. leaves G' unchanged;
- if l_i is a Series(i,j,k) reduction, then either l_i'=Tail(j) or l_i' is the empty reduction.

□

Property 5

A simple reducible graph G (i.e. without parallel arcs) contains at most $2n-3$ arcs.

Proof

Observe that each reduction operation decreases the number of arcs exactly by one; moreover, an operation Parallel(i, j) can be applied to G only after an operation Series(i, k, j). Since after at most $n-2$ series or tail reductions G is reduced to a single arc, at most $2(n-2)+1$ arcs can be deleted, i.e. we have at most $2n-3$ arcs in G. □

We say that a graph has the *reducibility property* if it is reducible and we say that it has the *series-parallel property* if it is series-parallel. In order to give a characterization of the reducibility and series-parallel properties, we consider the class Φ of graph properties which are determined by the *3-connected components*, described by Asano [1]. We report the following characterization, stated in terms of *2-connected components* [14], of the properties in the class Φ. A graph property μ is in Φ if the following conditions hold:

i) a graph G has the property μ if and only if each 2-connected component of G has the property μ;

ii) for 2-connected components, μ is determined by the 3-connected components.

It is easy to see that a 2-connected component of a reducible graph is series-parallel, since it can be reduced to a single arc without applying tail reductions; actually, a graph is reducible if and only if its 2-connected components are series parallel, in other words, for 2-connected components reducibility coincides with series-parallel. It is known that the series-parallel property is in Φ [1], hence it follows that the reducibility property is in Φ.

An algorithm that recognizes reducible graphs can be easily obtained as follows: given an input graph G, repeatedly apply reductions to G as far as possible; G is reducible if and only if the resulting reduced graph contains a single node. We will now show that this algorithm can be implemented in linear O(m) time; without loss of generality, we assume that G is simple.

Graph G is represented by adjacency lists: for each node i, $AD(i)$ contains the arcs incident with i. Each $AD(i)$ is implemented as a doubly linked list, whose elements are the arcs of G; each arc $e = (i,j)$ has pointers to the nodes i and j. It is easy to see that with the above data structures, each deletion and insertion of one arc in $AD(i)$ takes O(1) time.

Reductions are performed as deletion and insertion of arcs; Tail(i) deletes arc (i, j) from $AD(j)$; Series(j, i, k) deletes arcs (i, j) and (i,h) from $AD(h)$ and $AD(j)$, inserting a new arc (j,h); Parallel(i, j) deletes two parallel edges a and a' from $AD(i)$ and $AD(j)$, inserting a new arc (i, j).

At the beginning of the algorithm, each list $AD(i)$ is checked to find nodes with degree two or one; this takes an overall O(m) time. Then, each time a list $AD(i)$ is modified, we check $AD(i)$ to search for another reduction: if $|AD(i)|$ is one or two, then a tail or series reduction is applicable, otherwise, if there exist two parallel arcs a, a' in $AD(i)$ then a parallel reduction is applicable. Note that it is not necessary to scan the whole list $AD(i)$, but it suffices to scan the first three arcs in $AD(i)$. In fact, if no parallel arcs are found, there exist three non-parallel arcs in $AD(i)$; if G is reducible, at least one $AD(i)$ has less than three non-parallel arcs.

Observe that each $AD(i)$ can be checked in O(1) time, and that at most two lists must be checked after each reduction. Since at most m reductions can be applied to G, the recognition algorithm can be implemented with an overall O(m) complexity.

It is not difficult to devise an algorithm that, given a non-simple graph G, performs all possible parallel reductions in time O(m). It follows that we can recognize non-simple reducible graphs in linear time. Moreover, if a graph G is not reducible, we can perform all the possible reductions for G in linear time.

3. A solution algorithm for QSAP on reducible graphs

Consider the undirected graph $G(N,A)$ where the set of nodes N is $\{1,\ldots,n\}$ (each node represents a process) and the set of arcs A is determined by the coefficients f_{ij}, that is $A=\{(i,j): f_{ij}>0 \text{ or } f_{ji}>0\}$; let $m=|A|$. Further on, we assume that G is connected; in fact, if G is not connected, one independent QSAP for each connected component of G can be identified.

When the graph G is reducible, the corresponding QSAP can be solved in polynomial time. In order to carry out the computation of the QSAP optimal solution, we introduce some labels associated to the nodes and the arcs of G. In particular we will associate the labels u_{ir} $\forall r \in M$ to each node $i \in N$, and the labels v_{irjs} $\forall r, s \in M$ to each arc $(i,j) \in A$. Initially these labels are set as follows:

$u_{ir} = e_{ir}$, $\forall i \in N$, $\forall r \in M$,

$v_{irjs} = f_{ij}d_{rs} + f_{ji}d_{sr}$, $\forall\ i, j \in N$, $\forall r, s \in M$.

Note that initially the labels associated to each node represent the set of all possible processor assignments for the related process, and their initial value is the execution time of the process on the different processors. The arc labels represent the set of all possible assignments for a pair of communicating processes; their initial value is the communication time.

Our solution method consists of updating the labels according to the reduction operations performed on G. At the end, when G has been reduced to a single node, the minimum label of that node gives the optimal solution value.

The label updating can be described as follows:

- *Tail reduction*

  Let $i \in N$ be a node of degree 1 and $(j,i) \in A$ be the arc connecting i to the graph G. Labels u_{jr} are modified as follows, for each $r \in M$:

  $$u_{jr} = u_{jr} + \min \{u_{is} + v_{isjr},\ s \in M\}. \qquad (3.1)$$

  In practice u_{jr} is modified in order to take into account the best possible assignment for i once j has been assigned to r. This operation can be carried out in $O(p^2)$ time.

- *Series reduction*

  Let $i \in N$ a node of degree two and let (i,j) and (i,l) be the two arcs incident with i. Labels v_{jrls} are set as follows, for each $r, s \in M$:

  $$v_{jrls} = \min \{v_{jrit} + v_{itls} + u_{it},\ t \in M\} \qquad (3.2)$$

  The v_{jrls} gives the best possible assignment for i once j and l have been assigned to r and s, respectively. This operation can be carried out in $O(p^3)$ time.

- *Parallel reduction*

  Let $a'=(i,j)$ and $a''=(i,j)$ be two parallel arcs of graph G. Let v'_{irjs} and v''_{irjs} be the labels associated to a' and a'' respectively. The labels v_{irjs} of the new arc that will substitute a' and a'' in G' are obtained as follows:

  $v_{irjs} = v'_{irjs} + v''_{irjs}$, $\forall\ r, s \in M$.

  This operation can be carried out in $O(p^2)$ time.

Since at most O(n) reduction operations are applied on a reducible graph G, the overall complexity of the transformations is O(np^3).

The computation above gives the value of the optimal solution. In order to obtain an optimal assignment ρ associated to that value an extra computation is needed. To this end, we store in a stack the local choices we make in each Series or Tail reduction operation. Note that we do not need to store any information when executing a Parallel reduction, since this operation does not perform any assignment.

The information stored in the elements of the stack is the following:

- *Tail reduction*
  let (i,j) and i the arc and the node eliminated by the reduction, and for each $r \in M$ denote by $i(r)$ the index $s \in M$ giving the minimum in (3.1). We put on the stack a label *Tail*, the nodes i and j, and the set $\{i(r): r=1,\ldots,p\}$;
- *Series reduction*
  let (i,j) be the new arc introduced by the reduction and let h be the eliminated node; for each pair r, $s \in M$, denote by $h(r,s)$ the index $t \in M$ giving the minimum in (3.2). We put on the stack a label *Series*, the nodes i, j and h, and the set $\{h(r,s): r,s=1,\ldots,p\}$.

At the end of the reduction, let u_{ir} be the minimum label of the remaining node i; we set $\rho(i) = r$. Then, we repeatedly remove elements from the stack and, according to the label *Tail* or *Series*, we perform the following operations:

Tail: let $r = \rho(j)$; set $\rho(i) = i(r)$;

Series : let $r = \rho(i)$ and $s = \rho(j)$; set $\rho(h) = h(r,s)$.

It is easy to see that $\rho(j)$ ($\rho(i)$ and $\rho(j)$, respectively) has been already assigned when a Tail (Series, respectively) reduction is considered, hence the above method correctly finds an optimal assignment ρ.

4. Enumerative methods for QSAP

In this section we consider the general case of QSAP, that is instances whose associated communication graph is not necessarily reducible. In particular, we investigate the application of our method for polynomially solvable cases in the framework of enumerative algorithms.

In the following we will refer to a general and rather simplified Branch and Bound schema. The original QSAP problem S is reduced to a sequence of subproblems, each one corresponding to a node of an *enumeration tree* T, whose root corresponds to S. The *leaves* of T correspond to subproblems whose optimal solution has been determined; each internal node u of T has p sons, obtained from the problem

corresponding to u by selecting a (not yet assigned) process i and assigning it to the p different processors. The set Q contains the subproblems to be examined, the algorithm terminates when Q is empty. No assumption is made on the way subproblems are selected from Q.

A standard bounding technique is used to *prune* the enumeration tree, thus ignoring subproblems whose optimal solution cannot improve on the value U of the current *incumbent* solution. Denote by $Z(\rho)$ the cost of the assignment ρ; our Branch and Bound schema (for minimization problems) can be written as follows:

Enumeration Schema

input: a QSAP problem S;

output: a minimum cost assignment;

Step 0 (initialization) Set $Q = \{S\}$; find a feasible assignment ρ_I; set $U = Z(\rho_I)$;

Step 1 (termination) if Q is empty, then return ρ_I; otherwise select and remove a subproblem P from Q;

Step 2 (bounding) generate a lower bound L for P; if $L \geq U$ then go to Step 1; generate a feasible solution ρ for P; if $Z(\rho) < U$ then set $U = Z(\rho)$, $\rho_I = \rho$; if $L = Z(\rho)$ then go to Step 1;

Step 3 (branching) Select a process i not yet assigned in P, add to Q the subproblems $P_1,\ldots,P_p$: in problem P_r the process i is assigned to processor r. Go to Step 1.

A *partial assignment* on a subset $D \subseteq N$ is a function φ that for each $i \in D$ gives the processor $\varphi(i)$ to which i is assigned. An *extension* of a partial assignment φ is an assignment ρ such that $\rho(i) = \varphi(i)$ for each $i \in D$. It is easy to see that each subproblem P is characterized by a different partial assignment φ, and that P corresponds to the problem of finding the extension ρ of φ with minimum cost. This problem can be formulated as a QSAP problem on the set $N' = N \setminus D$ of not yet assigned processes; in fact, the objective function can be written as:

$$\sum_{i,j \in D} f_{ij} d_{\varphi(i)\varphi(j)} + \sum_{i \in D} e_{i\varphi(i)} + \sum_{i,j \in N'} f_{ij} d_{\rho(i)\rho(j)} + \sum_{i \in N'} e_{i\rho(i)} + \sum_{i \in N', j \in D} f_{ij} d_{\rho(i)\varphi(j)} + \sum_{j \in N', i \in D} f_{ij} d_{\varphi(i)\rho(j)}.$$

Since the first two terms of the sum are constant, the objective function can be replaced by

$$\sum_{i,j \in N'} f_{ij} d_{\rho(i)\rho(j)} + \sum_{j \in N'} e'_{i\rho(i)}$$

where:

$$e'_{i\,r} = e_{i\,r} + \sum_{j \in D} f_{ij} d_{r\varphi(j)} + f_{ji} d_{\varphi(i)r}.$$

In practice, the cost of the communications between a process $i \in N'$ and those in D is included in the linear cost of i. Note that the communication graph corresponding to each subproblem P is the induced subgraph of the original communication graph G obtained by deleting the nodes in D.

4.1 Lower bounds

The existence of sharp and efficiently computable lower and upper bounds is a crucial part of enumerative algorithms. The problem has been widely studied in the literature; for example [8] and [9] present efficient methods for solving *relaxations* of particular formulations of QSAP.

Our algorithm for polynomially solvable cases can be applied to obtain bounds. This approach has been partially exploited in [12], where a preliminary experimental study is reported.

Let $G=(N,A)$ be a non reducible communication graph corresponding to a given QSAP problem; and let $G_r = (N, A_r)$ be a reducible subgraph of G. We can define a new problem *restricted* to graph G_r, in which only communication costs corresponding to arcs in A_r are considered; the objective function becomes:

$$\sum_{(i,j) \in A_r} f_{ij} d_{\rho(i)\rho(j)} + f_{ji} d_{\rho(j)\rho(i)} + \sum_{i \in N} e_{i\,\rho(i)}.$$

Assume that the quadratic costs corresponding to arcs in $A \setminus A_r$ are non-negative: it can be verified that the optimal solution Z_r of the problem restricted to G_r is a lower bound for the original problem. We call *subgraph bound* the value obtained in this way.

Consider the partition of the set of edges $A \setminus A_r$ in k subsets $A_1,\ldots,A_k$ such that each partial graph $G_l = (N, A_l)$, $1 \le l \le k$, is reducible, and the problems restricted to graphs G_l, in which linear costs are set to zero:

$$Z_l = \min \{ \sum_{(i,j) \in A_l} f_{ij} d_{\rho(i)\rho(j)} + f_{ji} d_{\rho(j)\rho(i)} : \rho \in \Pi \}.$$

In the light of the above decomposition, the optimal solution value of (1.1) can be written as:

$$Z = Z_r^* + \sum_{l}^{k} Z_l^*,$$

where Z_r^* and Z_l^*, $1 \le l \le k$, are the costs of the optimal solution of (1.1) in the problems defined on graphs G_r and G_l, $1 \le l \le k$. It is easy to see that $Z_r \le Z_r^*$ and $Z_l \le Z_l^*$, hence the sum:

$$L = Z_r + \sum_{l}^{k} Z_l \leq Z_r^* + \sum_{l}^{k} Z_l^* = Z$$

is a lower bound for the original problem. We call *partition bound* the value L obtained as above; it is easy to see that the restricted problems G_r and $G_l = (N, A_l)$, $1 \leq l \leq k$, can be solved with an overall $O(mp^3)$ complexity.

Note that the partition bound can be used also when the nonnegativity hypothesis of the quadratic costs is relaxed, while the subgraph bound can be used only if the quadratic costs are non-negative.

Now we address the problem of determining the reducible subgraph G_r. In order to obtain a sharper bound, it is conceivable to search for a subgraph with a large set of arcs; moreover, arcs (i,j) corresponding to processes that exchange a large amount of information should be preferred. Thus one should find a reducible subgraph $G_r = (N, A_r)$ with maximum weight $W(G_r)$, where

$$W(G_r) = \sum_{(i,j) \in A_r} f_{ij} + f_{ji}.$$

Consider the problem of finding the reducible subgraph with the maximum number of arcs. This problem can be formulated as *minimum edge deletion*, which is a problem of the following form: given a graph G, find the smallest set of arcs to be deleted to obtain a subgraph satisfying a given property π. In our case π is the reducibility property, which belongs to the class Φ of properties determined by the 3-connected components. It has been proven ([1]) that for the properties in Φ the corresponding minimum edge deletion problem is NP-Complete, even when G is restricted to be a planar graph.

Thus the reducible subgraph of maximum cardinality or maximum weight cannot be easily identified. This is not true, however, if we require that G_r is a tree; in fact, many efficient algorithms for finding a maximum spanning forest in a graph are known [14]. The resulting restricted problem can be solved in time $O(np^2)$. For the same reason in the partition bound one may think to restrict G_l, $1 \leq l \leq k$, to be trees. In this case the resulting complexity is $O(mp^2)$.

In practical applications, it is necessary to devise efficient heuristic algorithms to identify subgraphs with sufficiently large weight. It must be observed that the maximum weight subgraph G_r does not always give the best lower bound. Consider the problem whose communication graph is given in Fig. 5. The numbers near the arcs represent the weights $f_{ij} + f_{ji}$.

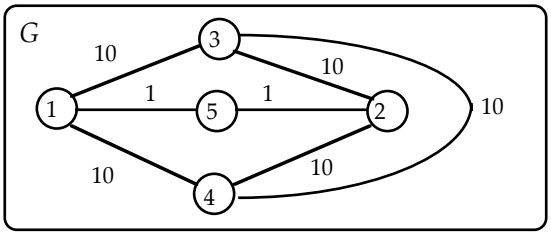

Fig. 5

We have p=2, d_{rs}=1, for each $r \neq s$, and d_{rr}=0 for each r; the execution costs are the following: e_{i1}=0, e_{i2}=1, $i \in \{1, 2, 3, 4\}$; e_{51}=10, e_{52}=0.

A maximum weight reducible subgraph can be obtained deleting arc (1,5); a lower bound $L = 1$ is obtained setting $\rho(i) = 1$, $i \in \{1, 2, 3, 4\}$, and $\rho(5) = 2$. If the arc (3,4) is deleted instead of (1,5) a bound $L = 2$ is obtained from the same assignment (which is the optimal assignment for the problem).

It is conceivable to require G_r to be *maximal*, i.e. that no arcs in $A \setminus A_r$ can be added to A_r obtaining a reducible graph. A trivial algorithm to find a maximal reducible subgraph has a O(nm) complexity; an interesting problem is the one of finding a maximal reducible subgraph in less than O(nm) time. Similar problems arise when a partition of the graph into k reducible subgraph is searched.

4.2 Branching selection rules

The solution methods for polynomially solvable cases of QSAP suggest some possible strategies for selecting the process to be assigned during each branching step. Suppose that the graph G_P corresponding to a subproblem P is reducible; it follows that P can be solved easily: actually, the optimal solution is obtained by computing the lower bound. It seems reasonable to adopt a *topological* selection strategy, where nodes are selected in order to obtain easily solvable subproblems as quickly as possible. This strategy would keep the enumeration tree T "flat", minimizing its *depth*, i.e. the length of the paths from the root to the leaves.

The problem of finding a selecting strategy that minimizes the (maximum) depth of T can be formulated as *node deletion problem*: given the graph G, delete a minimum size set of nodes so that the resulting induced graph has the reducibility property. In [11] it has been proved that node deletion problems are NP-Complete for all those graph properties μ that are hereditary on induced subgraph, i.e., if G has the property μ, then any induced subgraph does. From Property 4, we can conclude that the problem of finding an optimal selecting strategy is NP-Complete.

As a consequence, heuristic topological selecting strategies must be devised, such as selecting the node with higher degree. Note that such a strategy can be determined

at the beginning of the algorithm, finding a minimal set D of nodes to delete to obtain a reducible graph.
Once the set D has been detected, nodes can be selected in any order; actually, a topological strategy can be combined with many other selection strategies, e.g. selecting nodes in D with larger costs.

4.3 Branch & Shrink

We know that a subproblem P can be solved when the associated graph $G_P = (N_P, A_P)$ is reducible; if this is not the case, it may be possible to directly apply some reduction operations to G_P, obtaining a new "shrunken" graph $G_R = (N_R, A_R)$. Note that $N_R \subset N_P$, but in general $A_R \not\subseteq A_P$, since new arcs can be added to the graph applying the reduction operations. According to the performed reductions, the techniques defined in section 2 can be used to obtain new linear costs associated to nodes in N_R and new quadratic costs on arcs in G_R. The new problem P_R obtained in this way is equivalent to P; in particular, given any assignment $\rho_R : N_R \rightarrow M$ the optimal extension $\rho_P: N_P \rightarrow M$ of ρ_R can be obtained with the same technique used to find the optimal assignment for an easy solvable instance.
The above considerations suggest a "Branch and Shrink" approach, in which a subproblem P is replaced, whenever possible, by a reduced problem P_R obtained by applying to G_P all the possible reductions. In this approach some reduction operations, which may be repeatedly applied to the subgraphs of graph G_P during the Branch and Bound algorithm, will be applied only once and directly to G_P. This idea can save a considerable amount of computation, since repeated operations are avoided and smaller graphs are to be considered, e.g. when looking for reducible subgraphs.
In some cases, however, the shrinking can affect the quality of the subgraph lower bound. Consider again the graph G shown in Fig. 5; arcs (1,5) and (5,2) may be replaced by a single arc (1,2) by applying a series reduction. Then, a reducible subgraph of the resulting graph is obtained by deleting arc (1,2). As a result, a lower bound $L = 0$ is obtained from the assignment $\rho(i) = 1$, $i \in \{1, 2, 3, 4\}$: recall that a lower bound greater or equal to one can be obtained by selecting a maximal reducible graph of G.
Clearly, in order to avoid such pathological behaviour, an effective "Branch and Shrink" algorithm requires a careful implementation of the subgraph selection phase.

5. Numerical comparison of lower bounds

In the present section we will compare the lower bound presented in [12] with the lower bounds obtained by applying the results of section 4.1; in particular we will focus on the effectiveness of the partition technique and the use of reducible graphs.
In the following tables **Tree** and **Red** are the subgraph bounds which use a spanning tree and a maximal reducible subgraph, respectively; **Tree_P**, and **Red_P** are the partition bounds which use a decomposition into trees and reducible subgraphs, respectively. Problems with n=100 and m={400, 1000, 2500} have been considered. Each table entry contains the average value of the bound over a sample of 10 instances.
Tables 1 and 2 report the results for QSAP of type (1.1), where d_{rs} ($r \neq s$) represents the distance on a mesh of size 2×4 (p=8), 4×4 (p=16) and 4×8 (p=32), f_{ij} are integer and uniformly distributed in [1..10]. In table 1 the distance d_{rr}, r=1,…,p, is uniformly distributed in [0..1]; the linear costs e_{ir} are equal to $\delta\varepsilon_{ir}$, where $\delta = mp/4n$ and ε_{ir} are integer and uniformly distributed in [1..10]. This implies that, on average, the total linear cost is of the same order of the total quadratic cost. In table 2 d_{rr} ,r=1,…,p, is equal to the maximum distance; the linear costs e_{ir} are uniformly distributed in [1..10]. The choice of having nonzero distances d_{rr}, r=1,…,p, is suggested by the fact that if d_{rr}=0, r=1,…,p, the partition bound is equal to the subgraph bound. In fact for the problems on graphs G_l, $1 \leq l \leq k$, an optimal solution $Z_l = 0$ can be obtained by assigning all processes to the same processor.

m	*p*	Tree	Red	Tree_P	Red_P
400	8	1233.3	1360.4	1878.9	1962.2
400	16	1227.8	1378.7	1873.4	1980.5
400	32	1600.5	1859.1	2790.3	2973.0
1000	8	2723.8	2924.5	4352.3	4495.4
1000	16	2672.4	2911.7	4300.9	4482.6
1000	32	3378.8	3753.5	6406.9	6682.6
2500	8	6416.6	6698.1	10443.4	10653.8
2500	16	6344.2	6668.0	10371.0	10623.7
2500	32	8065.1	8600.8	15552.9	15958.2

table 1

m	*p*	Tree	Red	Tree_P	Red_P
400	8	435.8	573.8	1625.3	1692.5
400	16	391.5	531.7	1581.0	1650.4
400	32	444.4	646.7	2384.2	2495.6
1000	8	411.6	613.9	3721.0	3951.6
1000	16	370.3	578.7	3679.7	3916.4
1000	32	368.9	669.7	5684.2	6097.5
2500	8	411.2	703.7	8952.3	9808.0
2500	16	370.0	673.1	8911.1	9777.4
2500	32	358.5	777.3	13900.0	15356.2

table 2

The above tables show that **Red** dominates **Tree**, and the difference between the two bounds is usually larger for problems in table 2.

Moreover the partition technique is worth applying, in particular when quadratic costs dominate the linear ones (see table 2). Note that the partition is effective also when spanning trees are used; in fact, the relative difference between **Red_P** and **Tree_P** is usually smaller than that of **Red** and **Tree**.

Table 3 reports the results for problems of type (1.2), where both the quadratic and the linear costs are uniformly distributed in [0..10].

m	*p*	Tree	Red	Tree_P	Red_P
400	8	188.9	229.8	196.7	252.2
400	16	105.1	138.8	105.1	138.9
400	32	86.1	133.4	86.1	133.4
1000	8	190.9	263.7	218.2	394.2
1000	16	108.0	170.4	108.0	183.9
1000	32	85.7	172.1	85.7	191.6
2500	8	192.9	304.7	271.3	849.9
2500	16	106.1	199.2	106.1	326.4
2500	32	84.9	218.6	84.9	416.2

table 3

For these problems, the partition technique is sometimes less effective than in the previous cases, in particular when p is large. Nevertheless **Red_P** gives good improvements when m is large. On the contrary the improvement of **Tree_P** on **Tree** is often negligible. This can be explained by the fact that, if $q_{ijrs} = 0$ occurs with high probability, the tail operation is likely to leave node labels u_{ir} unchanged. Since

when trees are considered, only tail reductions are used, and most of the values Z_l are likely to be zero.
Remark that we reported for problems of large size in order to put in evidence the typical behavior of the bounds. Problems of such a large size can be considered quite difficult to solve (see for example [12]).

6. Conclusions and future work

We defined a polynomial solution method for a class of QSAP instances. This method can be exploited within a Branch an Bound algorithm to obtain lower bounds or selection strategies; it also suggests a different enumerative approach, based on graph shrinking operations.
It must be remarked that the partition bound is independent of the signs of the costs and can be applied to different formulations of QSAP; this is not true, for example, for other lower bounds proposed in the literature ([8], [9], [12]). The preliminary computational results show that the bounds used in [12] can be quite poor for some classes of problems.
In this paper we have discussed in detail the implementation of our method and its use in enumerative algorithms. In particular, we have pointed out that finding "optimal" lower bounds or selection strategies is a hard problem. A more extensive computational experience is needed in order to assess the effectiveness of the proposed approaches and how they compare with other more traditional methods.
During the Branch and Bound computation, an interesting problem is the one of finding reducible subgraphs efficiently, in particular when they are required to be maximal. For this problem the use of reoptimization techniques may be exploited successfully, as it happens for the Maximum Spanning Tree problem.

References

[1] Asano, T., "An application of duality to edge-deletion problems" SIAM Journal on Computing, 1986. 16(2): pp. 312-331.

[2] Berge, "Graphs and Hypergraphs" 1973, North Holland.

[3] Bokhari, S.H., "Assignment problems in parallel and distributed computing" Boston: Kluwer Academic Publishers.

[4] Bokhari, S.H., "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system" IEEE Transactions on Software Engeneering, 1981, SE-7: pp. 583-589.

[5] Chretienne, P., "A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints" European Journal of Operational Research, 1989, 43, pp. 225-230.

[6] Dixit, V.V. and D.I. Moldovan, "The allocation problem in parallel production systems". Journal of Parallel and Distributed Computing" 1990, 8: pp. 20-29.

[7] Duffin, R.J., "Topology of series-parallel networks" J. Math Anal. Appl., 1965, 10: pp. 303-318.

[8] Gallo, G. and B. Simeone, "Optimal grouping of researchers into departments" Ricerca Operativa, 1993, 57: pp. 45-69.

[9] Gallo, G., E.M. Tomasin, and A.M. Sorato, "Lower bounds for the quadratic semi-assignment problem" 1986, Rutgers University, New Brunswick, NJ 08903.

[10] Hansen, P. and K. Lih, "Improved algorithms for partitioning problems in parallel, pipelined and distributed computing" 1989, Rutcor, Rutgers University, New Brunswick.

[11] Lewis, J.M. and M. Yannakakis, "The node-deletion problem for hereditary properties is NP-complete" Journal of Computer and System Sciences, 1980, 10(2): pp. 219-230.

[12] Magirou, V.F. and J.Z. Milis, "An algorithm for the multiprocessor assignment problem". Operations Research Letters, 1989. 8: pp. 351-356.

[13] Sahni, S. and T. Gonzalez, "P-complete Approximation Problems" ACM Journal, 1976. (23): pp. 555-565.

[14] Tarjan, R.E., "Data Structures and Network Algorithms" 1983, SIAM Publications.

[15] Towsley, D., "Allocating programs containing branches and loops within a multiple processor system" IEEE Transactions on Software Engeneering, 1986, SE-12(10), pp. 1018-1024.

[16] Valdes, J., E.L. Lawler, and R.E. Tarjan, "The recognition of Series Parallel digraphs" SIAM Journal on Computing, 1982, 11(2): pp. 299-313.