The Quorum Deployment Problem
(Extended Abstract)

Seth Gilbert[1] and Grzegorz Malewicz[2]

[1] Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
Cambridge, MA
sethg@mit.edu

[2] University of Alabama
Computer Science Department
Tuscaloosa, AL
greg@cs.ua.edu

Abstract. Quorum systems are commonly used to maintain the consistency of replicated data in a distributed system. Much research has been devoted to developing quorum systems with good theoretical properties, such as fault tolerance and high availability. However, even given a theoretically good quorum system, it is not obvious how to efficiently deploy such a system in a real network. This paper introduces a new combinatorial optimization problem, the *Quorum Deployment Problem*, and studies its complexity. We demonstrate that it is NP-hard to approximate the Quorum Deployment Problem within any factor of n^δ, where n is the number of nodes in the distributed network and $\delta > 0$. The problem is NP-hard in even the simplest possible distributed network: a one-dimensional line with metric cost. We begin to study algorithms for variants of the problem. Some variants can be solved optimally in polynomial time and some NP-hard variants can be approximated to within a constant factor.

Keywords. quorum systems, combinatorial optimization, fault-tolerance

1 Introduction

The most common technique for ensuring fault-tolerance in a distributed system is replication: the data or code is replicated at a large number of nodes in the network, thus ensuring that no small number of failures can derail the computation. The primary difficulty with this approach is ensuring the consistency of replicas, without increasing the cost of accessing the data too much. There is a fundamental trade-off between the fault-tolerance of the data and the cost of maintaining consistency.

Quorum systems have long been used (e.g., [10, 25, 8, 13]) to solve the problem of replica consistency. A *quorum*, q, is a set of nodes in the network, and a *quorum system* is a set of quorums, Q, such that every two quorums in Q share at least one node. That is, given two quorums, $q, q' \in Q$, there exists some node $i \in q \cap q'$; the intersection of these two quorums is non-empty.

In order to ensure the consistency of the data, when a node chooses to modify the data, it notifies some quorum, say, $q \in Q$, of the modification; when a node wants to access the data, it contacts some quorum, say, $q' \in Q$. Since the two quorums, q and q', intersect at some node, we can be sure that the read operation that accesses the data learns about the earlier modification. Variations on this technique are frequently used to implement data replication (e.g., [2, 4, 1, 6]). For example, Attiya et al. use this technique to construct a read/write shared memory ([3]), and

* This work is supported by MURI–AFOSR SA2796PO 1-0000243658, USAF–AFRL #FA9550-04-1-0121, NSF Grant CCR-0121277, NSF-Texas Engineering Experiment Station Grant 64961-CS, and DARPA F33615-01-C-1896.

this is later extended to construct a reconfigurable read/write shared memory ([15, 11]). A similar technique has been used for mutual exclusion protocols (e.g., [8, 16]) and secure access protocols (e.g., [18]).

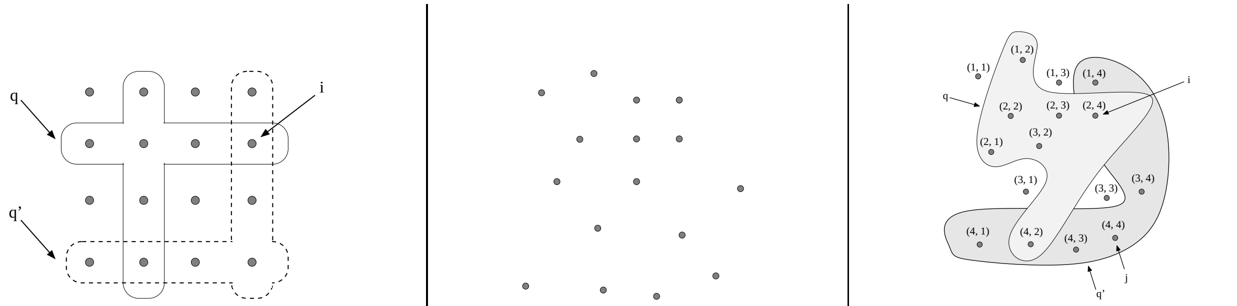
Much of the original work on quorum systems assumes that each quorum consists of a majority of the nodes in the network. In this way, the intersection property is immediately guaranteed, and optimal fault-tolerance is achieved. (See, for example, [28, 10, 29].) More recently, however, there has been much research developing more complicated quorum systems with a variety of interesting properties, such as improved availability, faster responses, and more flexibility to respond to dynamic systems. (See, for example, [5, 22, 17, 20, 19].)

Typically, an algorithm designer first constructs quorums with these types of good properties, and only then decides which network node will use which quorum so as to achieve low cost of network communication. Tsuchiya et al. [27] and Fu [7], on the other hand, have taken a different approach; their algorithms begin with a network, and determine a quorum system that is optimized under certain performance metrics. Unfortunately, the resulting quorum systems do not necessarily guarantee good fault tolerance, availability, etc. By first designing the quorum system, and then determining a good deployment, it seems possible to obtain both good network performance as well as good quorum system properties.

Let us illustrate this process in an example. Consider the quorum system in Figure 1(a) (originally described in [5]). The nodes in the network are arranged in a grid with $\sqrt{n}$ nodes in each row and column. Each quorum consists of one row and one column. Any two quorums, then, intersect at two nodes; for example, in Figure 1(a) quorums q and q' intersect at node i. Figure 1(b) represents an arbitrary network embedded in a two-dimensional plane in which the cost of communication between any two nodes is proportional to the distance between the nodes. In order to use the quorum system, each node in the real network must be mapped to a node in the grid, as in Figure 1(c). Then, each node chooses one of the quorums to use. For example, node i might choose to use quorum q, while node j might choose to use quorum q'. In an optimal world, each node is close to all the nodes in the quorum that is chooses.

If the quorum system is badly deployed, the cost of maintaining consistent replicas may be prohibitively expensive. It turns out that for completely natural quorum systems – and real world networks – the difference between an optimal deployment and a sub-optimal deployment can be quite large. In fact, we can show that for *every* non-trivial quorum system, there is some network in which an optimal deployment is much better than a bad deployment. If there are two nodes connected by an expensive communication link (for example, the network is occasionally partitioned), a sub-optimal deployment may require the nodes to communicate while an optimal deployment may not.

In this paper, we introduce the *Quorum Deployment Problem*, the problem of using a quorum system optimally. We assume that the set of quorums is fixed, and that the cost of sending a message between any two nodes is known in advance. The cost for some node, i, of using a quorum system is defined to be the cost of sending a message to every node in some quorum. Our goal is to determine the mapping from the real nodes in the network to the abstract nodes in the quorum specification, and the choice of which quorum each node should use during an operation. We present the problem more formally in Section 2.



(a) Grid quorum system

(b) Network of real nodes

(c) Mapping of real nodes to quorum system

Fig. 1. Figure 1(a) represents an abstract quorum system of 16 nodes, where q and q' are two possible quorums, and i is a node in the intersection. Figure 1(b) is an example of a network of nodes, embedded in a two-dimensional plane; communication time between two nodes is proportional to their distance. Figure 1(c) is a mapping of the real nodes in the network onto the abstract nodes in the quorum system.

Summary of Results

Our goal in this paper is to determine when the Quorum Deployment Problem can and cannot be efficiently solved. We first notice that a more constrained version of the problem, the *Partial Deployment Problem*, is solvable in polynomial time (see Section 3). The general version of the Quorum Deployment Problem, though, is quite hard. Even in the simplest possible distributed network – where the nodes are arranged in a line – the problem is NP-hard.

The natural question, then, is whether it is possible to determine an *approximately* optimal deployment. We show in Section 4 that it is NP-hard to approximate an optimal deployment within any constant factor. In fact, it is hard to approximate an optimal deployment within any factor of n^δ for any $\delta > 0$, where n is the number of nodes in the network.

Finally, in Section 5, we explore special cases (that are still NP-hard) in which the problem can be approximately solved, and in Section 6, we conclude and discuss future work.

2 The Quorum Deployment Problem

In this section, we formally define the *Quorum Deployment Problem*. The goal of the Quorum Deployment Problem is to determine, given a quorum system and a distributed network, how best to make use of that quorum system.

More formally, assume we are given a distributed network consisting of n nodes, connected by a message-passing network. We are given an n by n matrix, C, that specifies the cost of sending a message from node i to node j: $C_{i,j}$ is assumed to be the latency of the network connecting i and j. In this paper, we assume that the communication network is fixed. Anytime the network changes, the deployment must be recalculated, resulting in a quorum reconfiguration.

We are also given a quorum system, Q. For concreteness, we assume that Q consists of exactly n quorums, one for each node in the network. While quorum systems with more – and fewer – quorums may be interesting, we discover that the problem is quite hard even with this restriction. We assume that the quorum system is specified as an n by n matrix, where the columns represent the nodes in the quorums and the rows represent the quorums. Each entry in the matrix is either a 0 or a 1. Quorum p contains node j if (and only if) $Q_{p,j} = 1$. (See Figure 2(b) for an example of a quorum system in matrix form.)

Recall that the original notion of a quorum system assumes that every pair of quorums intersect. Occasionally in this paper, we relax this restriction, and allow the matrix Q to contain quorums that do not share a node. It turns out that the relaxed version of the problem is polynomially equivalent to the strict version of the problem.

A quorum deployment, then consists of two components. First, recall that each column in the quorum matrix represents a node; therefore each column in the quorum matrix must be assigned to a node in the network. This determines which real nodes are in each quorum. If node i is assigned to column j, then $Q_{p,j}$ determines whether node i is in quorum p. (Recall that each row of Q represents a quorum.)

Second, each node is assigned a quorum to use. Typically when using a quorum system, a node performing an operation must send a message to every node in some quorum, or receive a message from every node in some quorum. If, for example, node i is assigned quorum p, then whenever an operation occurs at node i, it first attempts to contact quorum p. If this fails (due to the failure of nodes in quorum p, for example), then node i may contact other quorums. (It is a separate – and harder – problem to determine a sequence of quorums to contact.) In this paper, we attempt to optimize for the common case, where quorum p has not failed. For each node i, the cost of the deployment is determined by the cost of accessing each node in its assigned quorum. For example, if node i is assigned quorum p, then the cost of the quorum deployment for i is:

$$\sum_{j \in p} C_{i,j}$$

We express each of the two components of quorum deployment as a permutation on $[1, n]$. We refer to the first component, the assignment of a node to a column in the quorum matrix, as the permutation β. That is, node i is assigned to column j if $\beta(i) = j$. Therefore, if node i is assigned quorum p, then the cost of the quorum deployment for i is:

$$\sum_{j=1}^{n} C_{i,j} \cdot Q_{p,\beta(j)}$$

The first term determines the cost of accessing node j, and the second term determines whether node j is in quorum p: the term $Q_{p,\beta(j)}$ is 1 if the column assigned to j is part of quorum p.

We refer to the second component of the quorum deployment, the assignment of a quorum to each node, as the permutation α. Node i is assigned quorum p if $\alpha(i) = p$. Therefore, the cost of quorum deployment for node i is:

$$\sum_{j=1}^{n} C_{i,j} \cdot Q_{\alpha(i),\beta(j)}$$

The total cost of a quorum deployment is the total cost of deployment for all the nodes in the network. Therefore, the total cost of deployment, $D(C, Q, \alpha, \beta)$ is:

$$D(C, Q, \alpha, \beta) = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{i,j} \cdot Q_{\alpha(i),\beta(j)}$$

Our goal is to minimize this cost: given matrices C and Q, find two permutations α and β on $\{1, \ldots, n\}$ that minimize $D(C, Q, \alpha, \beta)$ across all possible choices for α and β. We call this optimization problem the Quorum Deployment Problem.

Throughout the paper, we occasionally consider variants and restricted versions of the Quorum Deployment Problem. We describe these in more detail as they arise. The following is a brief preview of the variants:

- *Relaxed Quorum Deployment*: In this variant, the "quorums" are not required to intersect[3]. We may at times refer to the original problem as the *strict* deployment problem.
- *Partial Quorum Deployment*: In this variant, one of the two permutations, α or β, is given in advance as part of the problem instance.
- *Linear Quorum Deployment*: In this variant, the communication network is restricted to be a linear network. That is, all the nodes in the distributed network are embedded on a line.
- *Metric Cost Quorum Deployment*: In this variant, the cost matrix defines a metric. In particular, the distances between the nodes satisfies the triangle inequality.

3 Partial Quorum Deployment

We first consider the restricted problem of *Partial Quorum Deployment*. In the general Quorum Deployment Problem, we are given a quorum, Q, and a distributed network, C, and our goal is to determine a deployment, $\langle \alpha, \beta \rangle$, that has optimal cost. In the *Partial Quorum Deployment* problem, we assume that one of the two permutations in the deployment is fixed. That is, we assume that either α or β is given.

In one case, the permutation α may be fixed in advance. For example, α may be fixed as the identity: node 1 uses quorum 1, node 2 uses quorum 2, etc. The goal is to determine the permutation β, the assignment of nodes to the columns of the quorum matrix.

In the second case, the permutation β is fixed in advance. The goal, then, is to determine the permutation α, the assignment of which quorum each node should use.

Both cases of the Partial Deployment Problem can be reduced to the *Assignment Problem*, which has been well studied and can be solved in polynomial time (see, for example, [24]).

In the *Assignment Problem*, we are given a weighted bipartite graph, consisting of $2n$ nodes – n left nodes, L, and n right nodes, R – and a weight function $w_{i,j} \geq 0$ for all $i \in L$ and $j \in R$. The goal is to choose a matching consisting of n edges with minimum weight.

Theorem 1. *Given an instance of the Partial Deployment Problem, consisting of C, Q, and α, we can determine an instance of the Assignment Problem (in $O(n^2)$ time) where the solution to the Assignment Problem is the permutation β that minimizes the cost of the deployment. The same holds if the Partial Deployment Problem is specified to include β; the solution to the resulting Assignment Problem is the permutation α that minimizes the cost of the deployment.*

Proof. Assume that the permutation α is given. We construct a bipartite graph for the Assignment Problem where the left nodes, L, represent the nodes and the right nodes, R, repesent the columns in the quorum matrix, Q. The weight of an edge connecting $i \in L$ and $j \in R$ is the cost of assigning

[3] In this case, referring to the sets as "quorums" is a misuse of terminology, since the defining features of a set of quorums is that they intersect. For simplicity, however, we continue to use this term.

$$\begin{pmatrix} \begin{array}{cccc|cccc|c} n^x & n^x & n^x & n^x & n^x & 1 & 1 & n^x & 1 \\ n^x & n^x & n^x & n^x & 1 & n^x & n^x & 1 & 1 \\ n^x & n^x & n^x & n^x & 1 & n^x & 1 & 1 & 1 \\ n^x & n^x & n^x & n^x & n^x & 1 & n^x & 1 & 1 \\ \hline n^x & 1 & 1 & n^x & n^x & n^x & n^x & n^x & 1 \\ 1 & n^x & n^x & 1 & n^x & n^x & n^x & n^x & 1 \\ 1 & n^x & 1 & n^x & n^x & n^x & n^x & n^x & 1 \\ n^x & 1 & 1 & 1 & n^x & n^x & n^x & n^x & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \end{pmatrix}$$

(a) Cost Matrix, $Cost(G, k)$

$$\begin{pmatrix} \begin{array}{cc|cccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \end{pmatrix}$$

(b) Quorum Matrix, $Quorums(G, k)$

Fig. 2. An example of a reduction from the Balanced Complete Bipartite Subgraph problem in Figure 3 to the Quorum Deployment Problem.

i to column j in Q. That is:

$$w_{i,j} = \sum_{\ell=1}^{n} C_{\ell,i} \cdot Q_{\alpha(\ell),j} \ .$$

The Assignment Problem results in a permutation that minimizes the cost of the weights. The resulting permutation minimizes the cost of the quorum deployment.

Equivalently, if the permutation β is given, the left nodes in the bipartite graph represent the nodes and the right nodes represent the quorums; the weight of an edge represents the cost of a node using a given quorum. In this case:

$$w_{i,j} = \sum_{\ell=1}^{n} C_{i,\ell} \cdot Q_{j,\beta(\ell)} \ .$$

Again, the Assignment Problem minimizes the weights, resulting in a permutation that minimizes the cost of the quorum deployment. $\square$

4 Hardness of the Quorum Deployment Problem

While the Partial Deployment Problem is readily solvable, the general Quorum Deployment Problem is quite hard. In this section, we first show in Section 4 that it is NP-hard to approximate the general Quorum Deployment Problem within *any* constant factor. In fact, for any $\delta > 0$, it is hard to approximate within a factor of n^{δ}, where n is the number of nodes in the network. We then show that another variant, the Metric Cost Deployment problem, is NP-hard, and that the relaxed version (where the quorums are not required to intersect) is also NP-hard to approximate.

Hardness of Approximation

Our main hardness result is derived from a gap-creating reduction from the Balanced Complete Bipartite Subgraph (BCBS) Problem (see [9] for a statement of the problem, and [21] for recent results). In this problem, we are given a bipartite graph, $G = (V, E)$, consisting of left nodes, L, and right nodes, R. We are also given a constant, k. The goal is to find a balanced complete bipartite subgraph of size $2k$, with k left nodes and k right nodes.

Throughout this section, we use the bipartite graph in Figure 3 as an example. Notice that this graph has a balanced, complete subgraph of size two, consisting of nodes 2 and 3 on the left (in L) and nodes 5 and 8 on the right (in R). However, there is no such subgraph of size three.

In our reduction, we produce an instance of the Quorum Deployment Problem that has an efficient deployment if and only if the graph G contains a balanced complete bipartite subgraph of size k.

First, we define the reduction, $Cost(G, k) = C$ and $Quorums(G, k) = Q$, that transforms an instance of the BCBS problem into an instance of the Quorum Deployment Problem. We choose $n = |V| + 1$. The first $n-1$ columns encode the original BCBS problem; the last column ensures that all the quorums intersect.

The cost matrix, C, is related to the "complement" of the incidence matrix for the graph, G: each edge in the matrix G results in a cheap link in the matrix C, while two disconnected nodes in G are connected by an expensive link in the matrix C. For the purposes of the reduction, we fix x so that n^x is sufficiently large. The size of x depends on the desired value of δ. (That is, $x = O(\delta)$.) Formally:

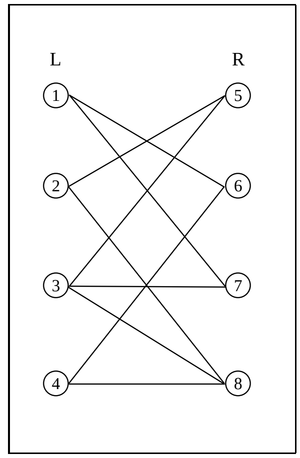


Fig. 3. Example instance of the Balanced Complete Bipartite Subgraph problem, where $k = 2$.

$$Cost(G,k)_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \text{ and } i,j < n \\ n^x & \text{if } (i,j) \notin E \text{ and } i,j < n \\ 1 & \text{if } i = n \text{ or } j = n \end{cases}$$

Consider the example in Figure 2(a). The submatrix delimited by the first four rows and first four columns represents the edges between nodes in L. Notice that because there are no edges between nodes in L, all the entries are n^x. The submatrix delimited by rows five through eight and columns five through eight represents edges between nodes in R, and therefore consists only of entries n^x. The last row and the last column contain the value 1. The remaining entries represent the edges between nodes in L and nodes in R. For example, the entry at $(3, 5)$ represents the edge between node 3 and node 5. Observe that the cost matrix is symmetric.

The quorum matrix, Q, consists of k quorums containing the first k nodes, and the extra node, n. It also contains a single quorum that contains all the nodes. The rest of the quorums contain only node n. Formally:

$$Quorums(G,k)_{i,j} = \begin{cases} 1 & \text{if } i,j \le k \\ 1 & \text{if } i = n \\ 1 & \text{if } j = n \\ 0 & \text{otherwise} \end{cases}$$

Consider again the example in Figure 2(b). The first two rows and two columns contains the value 1, representing the complete bipartite graph of size two. The last row and the last column contain the value 1, as well.

We show that if the original bipartite graph contains a balanced, complete bipartite subgraph of size k, then the derived Quorum Deployment Problem has a small cost. Alternatively, if the original bipartite graph does not contain such a subgraph, then the derived Quorum Deployment Problem results in a high cost deployment. A full proof is contained in Appendix A.

Lemma 1. *Fix any $x > 1$. Let $G = (V, E)$ be a bipartite graph, and let $1 \leq k \leq |V|$. Let $C = Cost(G, k)$ and $Q = Quorums(Q, k)$. Then the following holds:*

$$(G, k) \in BCBS \;\Rightarrow\; \exists\alpha, \exists\beta,\; D(C, Q, \alpha, \beta) \leq n^2$$
$$(G, k) \notin BCBS \;\Rightarrow\; \forall\alpha, \forall\beta,\; D(C, Q, \alpha, \beta) > n^x$$

That is, if there is a size k balanced, complete, bipartite subgraph in G, then the minimum cost of the resulting deployment is less then or equal to n^2. If there is not *a size k balanced, complete, bipartite subgraph in G, then the minimum cost of the resulting deployment is greater than n^x.*

Proof (sketch). The proof consists of two parts. In the first, we assume that $(G, k) \in BCBS$. In the second, we assume that $(G, k) \notin BCBS$.

Case 1 – $(G, k) \in BCBS$: First, suppose that there is a balanced complete bipartite subgraph on $2k$ nodes in G. We determine a deployment, (α, β) that has a small cost. Let $L' \subseteq L$ be the left partition of the subgraph and $R' \subseteq R$ the right partition of the subgraph. Choose α to map the nodes in L' to the first k rows, and choose β to map the nodes in R' to the first k columns. Node n is mapped to row n and column n. Then each of the quorum entries in the first k rows and k columns is mapped to one of the edges in the complete bipartite subgraph, and as a result, has cost 1. Each of the quorum entries in row n and column n is mapped to an entry in the cost matrix of cost 1. Therefore, the total cost of the deployment is $k^2 + 2n - 1 \leq n^2$, as desired.

Case 2 – $(G, k) \notin BCBS$: On the other hand, suppose that there is no complete bipartite subgraph on $2k$ nodes in G. We shall see that any deployment has cost larger than n^x. In particular, every deployment must include at least one expensive edge. It is clear that node n can, without loss of generality, be mapped to row n and column n: given an optimal assignment where this is not the case, it is possible to permute the assignment so that this is the case, without increasing the cost. Then notice that if there is a deployment that does not include any entry of n^x, then this implies that there exists a complete bipartite subgraph of size k, which we assumed was not the case. □

We conclude that the Quorum Deployment Problem is hard to approximate:

Theorem 2. *For any $\delta > 0$, it is NP-hard to approximate the Quorum Deployment problem with factor n^δ.*

Hardness of Metric Cost Quorum Deployment

In the Metric Cost Quorum Deployment Problem, the cost matrix is restricted to be symmetric and satisfy the triangle inequality. In this case, the cost of i sending a message to j is the same as the cost of j sending a message to i, and the cost of sending a message from i to j is no larger than the cost of sending a message from i to k and from k to j. It is clear from the reduction in Lemma 1 that this version of the problem is NP-hard:

Theorem 3. *The Metric Cost Quorum Deployment Problem is NP-hard.*

Proof. We use the same reduction as in Lemma 1, except instead of constructing the matrix $Cost(G, k)$ by setting non-edge costs to n^x, we set non-edge costs to 2. The matrix immediately satisfies the requirements of a metric. The correctness follows by the same argument as in Lemma 1, where if $(G, k) \in BCBS$ then the optimal cost of deployment is $k^2 + 2n - 1$; otherwise, if $(G, k) \notin BCBS$, then cost of any deployment is at least $k^2 + 2n$. □

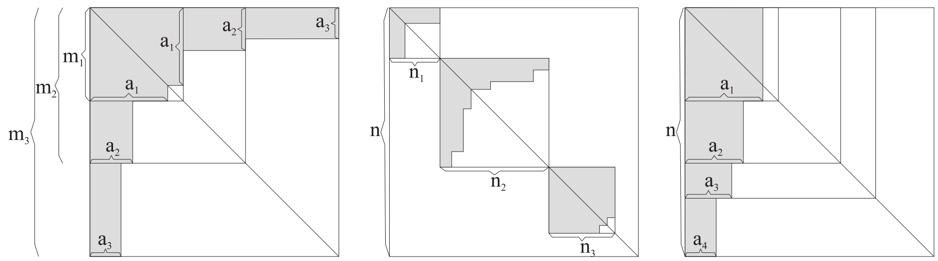


Fig. 4. Left: example of a hyperbola contained in $k = 3$ nested squares. Middle: example of a block diagonal hyperbolic quorum matrix with $p = 3$ hyperbolas with $k_1 = 1$, $k_2 = 3$ and $k_3 = 2$ nested squares respectively. Right: a quorum matrix composed of a part, called vertical telescope, of a single hyperbola. Note that any two quorums intersect in this matrix.

Hardness of Relaxed Metric Quorum Deployment

If we do not require that the "quorums" intersect, then we can show that such relaxed deployment problem is inapproximable even when the cost matrix is symmetric and satisfies the triangle inequality. The proof is inspired by the reduction from a strongly NP-complete 3-Partition Problem (see [9], SP15) to the Quadratic Assignment Problem (QAP) (see [9], ND43) given by Queyranne [23]. Our reduction extends the result of Queyranne. Since the deployment algorithm allows two degrees of freedom, α and β, as compared to QAP that has only one degree of freedom ($\alpha = \beta$ in QAP), we construct an instance of the deployment problem that reduces this flexibility, ensuring that when there is no 3-partition the cost of deployment is high. The proof is presented in Appendix B.

Theorem 4. *The Relaxed Metric Quorum Deployment Problem (with symmetric cost matrix that satisfies the triangle inequality and quorums that do not have to intersect) is NP-hard to approximate to within any constant factor.*

We note that the proof of this theorem implies that when the quorum matrix is a block diagonal matrix (ones inside blocks and zeros everywhere else) and the number of blocks can be as large as a polynomial fraction of n, then the deployment problem is inapproximable to within any constant factor. We also note that if the quorum matrix contains just one block, then it is NP-hard to optimally solve the problem. This follows from the proof of Theorem 3, where the bottom row and right column are trimmed from the matrices.

5 Approximation Algorithms for Metric Costs and Restricted Quorums

We have seen that if we allow arbitrary relaxed quorum matrix, then there is no constant factor approximation algorithm for the deployment problem even if we assume that the cost matrix is symmetric and satisfies the triangle inequality. It seems that the intricacy of the quorum matrix plays an important role in the ability to approximate the problem. Therefore, in this section, we establish a family of somewhat contrived quorum matrices that admit constant factor approximation for metric cost networks. Solving deployment optimally for this family, however, is still NP-hard.

We give a constant factor approximation algorithm for the Quorum Deployment Problem with a *block diagonal hyperbolic quorum matrix* and a symmetric cost matrix that satisfies the triangle inequality. The quorum matrix is composed of a constant number p of *hyperbolas* placed on the diagonal. Each hyperbola i is contained inside a constant number k_i of nested squares (see Figure 4, and a formal definition in the full version). The approximation factor is $c = 4 \cdot \max_{1 \le r \le p} k_r$. The algorithm runs in $O(n^{k_1+\ldots+k_p+3p})$ time.

Theorem 5. *There is a c-approximation algorithm for the Quorum Deployment Problem with a block diagonal hyperbolic quorum matrix and symmetric cost matrix that satisfies the triangle inequality, where $c = 4 \cdot \max_{1 \le r \le p} k_r$. The algorithm runs in $O(n^{k_1+\ldots+k_p+3p})$ time.*

The proof sketch that follows presents an overview of the approximation algorithm and our key observations. A detailed proof of the theorem is given in Appendix C. For convenience of the presentation, we specify the permutations α and β to rearrange rows and columns of the *cost* matrix rather than the *quorum* matrix. This of course yields an equivalent optimization problem.

Proof (sketch). Suppose for a moment that the quorum matrix has ones inside a submatrix $U \times U$, and zeros everywhere else. Let $m = |U|$. An optimal deployment will place some rows $\tilde{U'}$ and some columns $\tilde{V'}$ of the cost matrix inside $U \times U$. When we pick a row i and m columns V, that minimize the sum of costs at the intersection of the row and the columns, then by the triangle inequality and symmetricity of the cost matrix, we can conclude that the sum of costs inside the submatrix $V \times V$ is at most twice the cost of the optimal deployment. We notice that the conclusion is true even though the optimal deployment may have $\tilde{U'} \neq \tilde{V'}$, i.e., may indeed take advantage of two degrees of freedom to lower the cost. This observation extends the technique of Krumke et al. [14] developed for the Quadratic Assignment Problem where we would have $\tilde{U'} = \tilde{V'}$ (in QAP rows and columns are permuted in the same way).

Now suppose that the quorum matrix has the richer structure of a single hyperbola. Then an optimal deployment has extra ability to avoid high costs due to "holes" in the quorum matrix, as compared to the $U \times U$ case just discussed. We can show, however, how to effectively deal with these holes by appropriately rearranging rows and columns to "push" low costs to the areas occupied by the hyperbola, and leave high costs behind. The hyperbola is contained in k nested squares. The square h has size m_h by m_h and the hyperbola has thickness a_h at the edge of the square (cf. Figure 4). For each h, we can find a row i_h and m_h columns V_h that minimize the sum of costs at the intersection of this row and the columns. Since we have selected a row and columns that minimize the sum, clearly, the cost of any optimal deployment is at least $1/k \sum_{1 \le h \le k} a_h \sum_{j \in V_h} C_{i_h,j}$. This simplistic bound leaves too big a freedom in the choice of subsets V_h, and so the submatrices $V_h \times V_h$ would not be useful for approximation because the submatrices would not have to be nested. Recall that the k squares are nested in the optimal deployment, so we can still bound from below the cost of the deployment if we introduce a constraint that $V_1 \subset V_2 \subset \ldots \subset V_k$. With this constraint though, there are dependencies between V_h's. Hence we cannot perform the minimization of the sum $\sum_{j \in V_h} C_{i_h,j}$ across the choices of i_h and V_h independent from the minimization of the corresponding sums across other rows and other subsets of columns because we could get stuck in a local minimum. What we need to do instead, is to minimize the value of the entire bound across all possible choices under the constraint. We can find the nested subsets V_h and rows i_h that minimize the bound $\sum_{1 \le h \le k} a_h \sum_{j \in V_h} C_{i_h,j}$ using an appropriately adjusted polynomial time algorithm of Tokuyama and Nakano [26], in a fashion resembling the method used by Guttmann-Beck and Hassin [12]. After V_h's and i_h's have been found, we rearrange rows and columns. Using the triangle inequality and symmetricity of the cost matrix, we conclude that the costs inside submatrix $V_h \times V_h$ can be bounded from above by $2m_h$ times the costs at the intersection of row i_h and columns V_h. If we move the a_h lowest cost rows to the top part of the submatrix, then the sum of costs accumulated there is proportionally reduced, and so it is at most a a_h/m_h fraction of the sum of costs inside the entire submatrix. We rearrange rows of the submatrix $V_1 \times V_1$, then rows of $V_2 \times V_2$ and so on, and then columns. When rearrangements are done carefully, we can ensure that one rearrangement does not destroy the upper bounds on costs created by the prior rearrangements. After the rearrangements, the sum of costs inside the parabola will be at most $4 \sum_{1 \leq h \leq k} a_h \sum_{j \in V_h} C_{i_h,j}$. This completes approximation argument for a single parabola.

Finally, assume that the quorum matrix is a block diagonal hyperbolic quorum matrix composed of p hyperbolas. We modify the algorithm for finding nested subsets of columns, so that now the algorithm minimizes across p collections of nested subsets of columns. After we have found the collections, we apply, to each of the p collections of nested submatrices, the algorithm for rearranging rows and columns. This yields the desired approximation result and completes the proof. □

We contrast our approximation results with the inapproximability results from the previous section. When the number of hyperbolas can be as big as a polynomial fraction of n, then the deployment problem is inapproximable to within any constant factor, even when each hyperbola i is just a single square completely filled in with ones. However, we can approximate the problem to within a constant factor when the number of hyperbolas is constant, and even if each hyperbola is contained in more than one square.

6 Conclusions and Future Work

In this paper, we have introduced the Quorum Deployment problem, a natural problem that arises when attempting to efficiently replicate data. We have examined the complexity of a number of variants of the problem, showing that the Partial Deployment Problem can be solved in polynomial time, while the general Quorum Deployment Problem and the Relaxed Metric Deployment problem are inapproximable. Finally, we presented some special NP-hard cases in which the problem can be approximated and other cases that admit optimal polynomial time solution.

While many of the results presented in this paper are negative, we believe it is important to continue examining cases for which quorums may be efficiently deployed, as the problem has significant practical import. Most previous research has focused on developing quorum systems that have good robustness to various failure modes; future research should also take into account the difficulty of deploying the quorums. While we conjecture that most currently developed quorum systems (such as the grid quorum system) cannot be deployed efficiently, we would like to develop families of quorum systems that are both robust and can be deployed efficiently.

References

1. A. El Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. *Transactions on Database Systems*, 14(2):264–290, 1989.
2. D. Agrawal and A. El Abbadi. Resilient logical structures for efficient management of replicated data. Technical report, University of California Santa Barbara, 1992.
3. Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, 1995.
4. M. Bearden and R.P. Bianchini Jr. A fault-tolerant algorithm for decentralized on-line quorum adaptation. In *Proceedings of the 28th International Symposium on Fault-Tolerant Computing Systems*, Munich, Germany, 1998.
5. Shun Yan Cheung, Mostafa H. Ammar, and Mustaque Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *Knowledge and Data Engineering*, 4(6):582–592, 1992.
6. A. El Abbadi, D. Skeen, and F. Cristian. An efficient fault-tolerant protocol for replicated data management. In *Proc. of the 4th Symp. on Principles of Databases*, pages 215–228. ACM Press, 1985.

7. A. W. Fu. Delay-optimal quorum consensus for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(1):59–69, 1997.
8. Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.
9. M. R. Gary and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.
10. David K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh symposium on operating systems principles*, pages 150–162, 1979.
11. Seth Gilbert, Nancy Lynch, and Alex Shvartsman. RAMBO II:: Rapidly reconfigurable atomic memory for dynamic networks. In *Proc. of the Intl. Conference on Dependable Systems and Networks*, pages 259–269, June 2003.
12. N. Guttmann-Beck and R R. Hassin. Approximation algorithms for min-sum p-clustering. *Discrete Applied Mathematics*, 89(1-3):125–142, 1998.
13. Maurice Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, feb 1986.
14. S. O. Krumke, M. V. Marathe, H. Noltemeier, V. Radhakrishnan, S. S. Ravi, and D. J. Rosenkrantz. Compact location problems. *Theoretical Computer Science*, 181(2):379–404, 1997.
15. Nancy Lynch and Alex Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *Proc. of the 16th Intl. Symp. on Distributed Computing*, pages 173–190, 2002.
16. M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Tranactions on Computer Systems*, 3(2):145–159, 1985.
17. Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *Proceedings of the 29th Symposium on Theory of Computing*, pages 569–578, 1997.
18. Moni Naor and Udi Wieder. Access control and signatures via quorum secret sharing. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):909–922, 1998.
19. Moni Naor and Udi Wieder. Scalable and dynamic quorum systems. In *Twenty-Second ACM Symposium on Principles of Distributed Computing*, 2003.
20. Moni Naor and Avishai Wool. The load, capacity, and availability of quorums systems. *SIAM Journal on Computing*, 27(2):423–447, 1998.
21. R.J.P. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003.
22. D. Peleg and A. Wool. Crumbling walls: a class of high availability quorum systems. In *Proceedings of the 14th ACM Symposium on Principles of Distributed Computing*, pages 120–129, 1995.
23. M. Queyranne. Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. *Operations Research Letters*, 4(5):231–234, 1986.
24. Alexander Schrijver. *Combinatorial Optimization*, volume A, chapter 17. Springer, 2003.
25. Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *Transactions on Database Systems*, 4(2):180–209, 1979.
26. T. Tokuyama and J. Nakano. Geometric algorithms for the minimum cost assignment problem. *Random Structures and Algorithms*, 6(4):393–406, 1995.
27. Tatsuhiro Tsuchiya, Masatoshi Yamaguchi, and Tohru Kikun. Minimizing the maximum delay for reaching consensus in quorum-based mutual exclusion schemes. *IEEE Transactions on Parallel and Distributed Systems*, 10(4):337–345, 1999.
28. Eli Upfal and Avi Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34(1):116–127, 1987.
29. P.M.B. Vitányi and B. Awerbuch. Atomic shared register access by asynchronous hardware. In *Proceedings 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 233–243, New York, 1986. IEEE.

A Hardness of Approximation

Lemma 1. Fix any $x > 1$. Let $G = (V, E)$ be a bipartite graph, and let $1 \leq k \leq |V|$. Let $C = Cost(G, k)$ and $Q = Quorums(Q, k)$. Then the following holds:

$$
\begin{aligned}
(G, k) \in BCBS &\Rightarrow \exists \alpha, \exists \beta,\ D(C, Q, \alpha, \beta) \leq n^2 \\
(G, k) \notin BCBS &\Rightarrow \forall \alpha, \forall \beta,\ D(C, Q, \alpha, \beta) > n^x
\end{aligned}
$$

That is, if there is a size k balanced, complete, bipartite subgraph in G, then the minimum cost of the resulting deployment is less then or equal to n^2. If there is *not* a size k balanced, complete, bipartite subgraph in G, then the minimum cost of the resulting deployment is greater than n^x.

Proof. The proof consists of two parts. In the first, we assume that $(G, k) \in BCBS$. In the second, we assume that $(G, k) \notin BCBS$.

Case 1 – $(G, k) \in BCBS$: First, suppose that there is a balanced complete bipartite subgraph on $2k$ nodes in G. We determine a deployment, (α, β) that has a small cost.

Let $L' \subseteq L$ be the left partition of the subgraph and $R' \subseteq R$ the right partition of the subgraph. Notice that $|L'| = |R'| = k$, and $L' \cap R' = \emptyset$. Let $L' = \{\ell_1, \ldots, \ell_k\}$ and $R' = \{r_1, \ldots, r_k\}$. We define the deployment as follows:

$$
\alpha(i) = \begin{cases} j & i = \ell_j \\ n & i = n \\ \text{arbitrary} & \text{otherwise} \end{cases} \tag{1}
$$

$$
\beta(i) = \begin{cases} j & i = r_j \\ n & i = n \\ \text{arbitrary} & \text{otherwise} \end{cases} \tag{2}
$$

Notice that α assigns the first k columns of C to nodes in L', and β assigns the first k rows of C to nodes in R'.

We now determine the cost of this deployment. Recall that the first k rows and k columns of Q contain the value 1. Notice that when $\alpha(i) = 1$, $i = \ell_1$. Similarly, when $\alpha(j) = 1$, $j = r_1$. Therefore $i \in L'$ and $j \in R'$, which implies that there is an edge between nodes i and j. As a result, $C_{i,j} = 1$.

This holds for all $\alpha(i) \leq k$ and $\beta(j) \leq k$. Therefore the first k rows and k columns of Q contribute cost at most k^2.

The only other entries in Q with value 1 are row n and column n. However, node n is assigned to column n, and node n is assigned to row n. Since $C_{i,n} = 1$ for all i, and $C_{n,j} = 1$ for all j, row n and column n contribute cost $2n - 1$.

There are no other entries containing a 1 in Q. Therefore the total cost of the deployment is at most n^2.

Case 2 – $(G, k) \notin BCBS$: On the other hand, suppose that there is no complete bipartite subgraph on $2k$ nodes in G. We shall see that any deployment has cost larger than n^x. In particular, every deployment must include at least one expensive edge.

Let $\langle \alpha, \beta \rangle$ be the optimal deployment. We bound the cost of this deployment from below in two steps.

First, assume that $\alpha(n) \neq n$. In particular, assume that $\alpha(n) = a$, and $a \neq n$. Then node n uses the quorum designated by row a of Q. Some other node, b, is assigned to row n of Q. Consider the new deployment, $\langle \alpha', \beta \rangle$, where $\alpha(n) = n$ and $\alpha(b) = a$. We now argue that this new deployment has a cost no greater than that of the originaly deployment $\langle \alpha, \beta \rangle$.

Let A be the set of nodes in quorum a, that is, the set $\{j : Q_{a,j} = 1\}$. In the originaly deployment, the cost of node n accessing the nodes in A is $|A|$, since node n can access each node for cost 1. Let x be the cost of node b accessing quorum n.

In the revised deployment that includes α', node n accesses quorum n, which has cost exactly n. Let y be the cost of node b accessing nodes in the set A. Our goal is to show that $n + y \leq |A| + X$.

Notice that $y \leq x - (n - |A|)$. Since every node in A is also in quorum n, the cost y has to be less than the cost x. Since the smallest cost of accessing a node is 1, when node a accesses quorum n the cost must be at least $n - |A|$ more than the cost of accessing just the nodes in A. Therefore, we conclude that:

$$
\begin{aligned}
n + y &\leq n + (x - (n - |A|) \\
&\leq x + |A|
\end{aligned}
$$

Therefore the cost of the deployment containing α' is less than or equal to the cost of the original deployment. Therefore, without loss of generality we assume that $\alpha(n) = n$. By the same argument, we can assume without loss of generality that $\beta(n) = n$. Therefore row n and column n contribute exactly $2n - 1$ to the cost of the deployment.

Let L' be the set of rows mapped to the the first k rows. That is, $L' = \{i : 1 \leq \alpha(i) \leq k\}$. Let R' be the set of columns mapped to the first k columns. That is, $R' = \{j : 1 \leq \beta(j) \leq k\}$. These are the only entries of C that can contribute to the cost, aside from row n and column n.

If all the costs $C_{i,j} = 1$ for $i \in L'$ and $j \in R'$, then there is an edge between every node in L' and every node in R', thus creating a balanced complete bipartite subgraph of size k. Thus at least one of the cost $C_{i,j} = n^x$ for some $i \in L'$ and some $j \in R'$.

Thus the cost of the optimal deployment is at least $n^x + 2n - 1$, which is greater than n^x, as desired. $\square$

B Hardness of Relaxed Metric Quorum Deployment

Theorem 4. The Relaxed Metric Quorum Deployment Problem (with symmetric cost matrix that satisfies the triangle inequality and quorums that do not have to intersect) is not approximable to within any constant, unless P=NP.

Proof. Suppose that there is an r-approximation algorithm for the deployment problem. We will show a polynomial time reduction from 3-Partition to the deployment problem that creates a gap of at least $r + 1$. This will yield a contradiction when P$\neq$NP.

Take any instance of the 3-Partition Problem. We construct an instance of the restricted/relaxed Quorum Deployment Problem. There are Bk nodes in the network. Nodes are placed on a line and the line and distance between nodes induces the cost matrix. The nodes are grouped into k clusters, each containing B nodes. Consecutive clusters are separated by a "large" gap of $\Delta = r + 1$, and consecutive nodes inside a cluster are separated by a "small" gap of $\delta = 1/(kB^3)$. For each size, we construct as many quorums as the value of the size. The quorums will form a "clique", which will allow us to keep them "together" or else substantially increase the cost. For a given size s_i, we produce s_i equal quorums, each of cardinality s_i, each equal to $\{(s_1 + \ldots + s_{i-1} + 1), \ldots\ldots, (s_1 + \ldots + s_i)\}$. Note that the size of the instance of the Quorum Deployment Problem is polynomial with respect to k (recall that Bk is bounded by the value of a fixed polynomial of k, because 3-Partition is strongly NP-complete).

If there exists a 3-partition $S_1, \ldots, S_k$, then the minimal cost of deployment is "small". To see this, we consider the three sizes in S_h and assign the B quorums derived from these sizes to the cluster h. There are exactly B distinct elements in all quorums assigned to the cluster. At the same time, the cluster has exactly B nodes. Hence we can define a "local" mapping: the B quorum elements are arbitrarily mapped to the B nodes of the cluster. Since there are B quorums in a cluster, each quorum has at most B elements and the diameter of the cluster is at most $B\delta$, the total contribution of this cluster to the cost of the deployment is at most δB^3. Hence the total cost of deployment is at most $k\delta B^3$.

If there is no 3-partition, then the minimal cost of deployment must be "large". We consider three cases, always concluding that the minimal cost of deployment is Δ or more. Take a deployment that minimizes cost. If elements of a quorum are assigned to two different clusters, then cost of the deployment is at least Δ. So, let us assume that, for each quorum, the elements (or the only element, as a quorum may be a singleton) of the quorum are assigned to the same cluster. If a quorum is assigned to a cluster other than the cluster where the quorum elements are assigned to, then the cost of the deployment is at least Δ. Finally, assume that each quorum is assigned to the (only) cluster where the elements of the quorum are assigned. This, however, would imply that a 3-partition exists. Therefore, minimal cost of deployment is at least Δ.

To complete the inapproximability argument, it remains to notice that a sufficient gap has been created because of the choice of the value of δ and Δ. $\square$

C Approximation Algorithm for Metric Costs and Restricted Quorums

This section presents an approximation algorithm for the Quorum Deployment Problem with a block diagonal hyperbolic quorum matrix and a symmetric cost matrix that satisfies the triangle inequality.

Definitions

We define a *block diagonal hyperbolic quorum matrix*. Let the matrix Q be a block diagonal n by n matrix with zero entries outside of the blocks, and where each block is a *hyperbola* (see Figure 4 for an example). Specifically, the matrix is defined by several numbers. A sequence of $p \geq 1$ numbers $n_1, \ldots, n_p$ represents the sizes of the p blocks of the matrix. Each size has to be at least one, and the sizes must sum up to at most n. Each block i, that has size n_i, is a *hyperbola* defined by $k_i \geq 1$ numbers $0 < m_1^i < m_2^i < \ldots < m_{k_i}^i = n_i$, and k_i other numbers, such that $a_j^i \leq m_j^i$, for $1 \leq j \leq k_i$, $a_1^i > a_2^i > \ldots > a_{k_i}^i \geq 1$. The first hyperbola is defined as

$$H_1 = \left(\bigcup_{h=1}^{k_1} [a_h^1] \times [m_h^1] \right) \cup \left(\bigcup_{h=1}^{k_1} [m_h^1] \times [a_h^1] \right) .$$

The other hyperbolas are appropriately shifted to fit inside the corresponding blocks. Specifically, hyperbola i is defined as

$$H_i = \left(\bigcup_{h=1}^{k_i} [s_i + 1, s_i + a_h^i] \times [s_i + 1, s_i + m_h^i] \right) \cup \left(\bigcup_{h=1}^{k_i} [s_i + 1, s_i + m_h^i] \times [s_i + 1, s_i + a_h^i] \right) ,$$

where s_i is the *shift* equal to $\sum_{w=1}^{i-1} n_w$. The matrix Q is an n by n matrix that has 1 at any entry (y, x) that belongs to any of the p hyperbolas, and 0 everywhere else

$$Q_{y,x} = \begin{cases} 1, & (y, x) \in H_1 \cup \ldots \cup H_p , \\ 0, & (y, x) \notin H_1 \cup \ldots \cup H_p . \end{cases}$$

The matrix Q is called a block diagonal hyperbolic quorum matrix. Such matrix is a quorum matrix in the strict sense only when $p = 1$ and $n_1 = n$. When $p > 1$ or when $n_1 < n$, then not every quorum intersects with every quorum.

Algorithm and its analysis

We begin the approximation argument with a lemma that bounds from below the cost of an optimal deployment. We find a collection of sets of columns, some of which are nested, and a collection of rows. The costs located at the intersections of rows and columns, weighted appropriately, will yield a lower bound on the minimal cost of deployment. Our search algorithm uses an algorithm of Tokuyama and Nakano as a subprocedure.

Lemma 2. *Given any instance of the Quorum Deployment Problem with a block diagonal hyperbolic quorum matrix, there is an algorithm that selects rows i_h^r, not necessarily distinct, of the cost matrix C, $1 \leq r \leq p$, $1 \leq h \leq k_r$, and p collections in nested subsets of columns $V_1^r \subset V_2^r \subset \ldots \subset V_{k_r}^r \subseteq [n]$, $1 \leq r \leq p$, such that $|V_h^r| = a_h^r$, and that when $r \neq r'$, then $V_{k_r}^r$ does not intersect with $V_{k_{r'}}^{r'}$, so that*

$$\sum_{r=1}^{p} 1/k_r \sum_{h=1}^{k_r} a_h^r \sum_{d \in V_h^r} C_{i_h^r, d}$$

is a lower bound on the minimal cost of deployment for the instance of the problem. The algorithm runs in $O(n^{k_1 + \ldots + k_p + 3p})$ time.

Proof. Take an optimal deployment that somehow independently permutes rows and columns of the cost matrix. Let us focus on the hyperbola H_r, and the costs assigned to it. The n_r by n_r block gets assigned n_r rows and n_r columns. Let $\tilde{V}_h^r$ be the set of m_h^r leftmost columns of the cost matrix that are assigned to the block. Obviously, $\tilde{V}_1^r \subset \ldots \subset \tilde{V}_{k_r}^r$, and the sets $\tilde{V}_{k_r}^r$, for different r, are disjoint. The hyperbola H_r includes k_r "horizontal" rectangles that overlap. The rectangle $[s_r + 1, s_r + a_h^r] \times [s_r + 1, s_r + m_h^r]$ gets assigned columns $\tilde{V}_h^r$, where $1 \leq h \leq k_r$. Among the a_h^r rows assigned to the rectangle h, we can find a row $\tilde{i}_h^r$ that minimizes the sum $\sum_{d \in \tilde{V}_h^r} C_{\tilde{i}_h^r, d}$ of costs assigned to the row of the rectangle. The contribution, to the cost of the optimal deployment, of this rectangle is at least a_h^r times the sum. When we sum up the k_r lower bounds on costs of the rectangles, we establish a lower bound on k_r times the cost contributed by the hyperbola H_r. Hence the cost of the optimal deployment is bounded from below by an expression

$$min \geq \sum_{r=1}^{p} 1/k_r \left(\sum_{h=1}^{k_r} a_h^r \sum_{d \in \tilde{V}_h^r} C_{\tilde{i}_h^r, d} \right) .$$

We can bound the expression from below by minimizing across a larger set of choices: the choices of rows i_h^r, not necessarily distinct, and subsets of columns $V_1^r \subset \ldots \subset V_{k_r}^r$, such that $|V_h^r| = m_h^r$, and that the sets $V_{k_r}^r$, for different r, are disjoint, where indices range over $1 \leq r \leq p$, $1 \leq h \leq k_r$.

The selection of rows and subsets of columns to minimize the sum can be done by repeatedly applying the $O(n)$ deterministic algorithm of Tokuyama and Nakano [26]. The algorithm assumes that subsets are disjoint, while in our case some are nested and some are disjoint. We can comply with the assumptions of Tokuyama and Nakano by preprocessing the input. We consider all $n^{k_1+\ldots+k_p}$ selections for p tuples $(i_1^r, \ldots, i_{k_r}^r) \in [n]^{k_r}$, where r ranges from 1 to p. We take a table $(t_{i,j})$ of size $k_1 + \ldots + k_p$ by n, and partition it into rectangles of width n and height k_1, k_2 and so on, moving from the top to the bottom of the table. Then we copy rows $i_1^1, \ldots, i_{k_1}^1$ of the cost matrix into the top rectangle. For each row h of the rectangle, $h = k_1, (k_1 - 1), \ldots, 1$, we scale the row and add scaled costs of rows $h+1, \ldots, k_1$ of the rectangle i.e., $t_{h,j}$ becomes $\sum_{h \leq x \leq k_1} a_x^1 \cdot t_{x,j}$, for any $1 \leq j \leq n$. Observe that if we select a subset $A_h \subseteq [n]$ of cardinality $a_h^1 - a_{h-1}^1$ of costs inside each row h, different for different rows (i.e., $i \neq j$ implies $A_i \cap A_j = \emptyset$), and add the selected costs for all k_1 rows of the top rectangle, then the sum $\sum_{1 \leq h \leq k_1} \sum_{j \in A_h} t_{h,j}$ is equal to exactly the cost with nested subsets $\sum_{1 \leq h \leq k_1} a_h^1 \sum_{j \in V_h} C_{i_h, j}$, where $V_h = A_1 \cup \ldots \cup A_h$ has cardinality a_h^1 and $V_1 \subset \ldots \subset V_{k_1}$. The reverse relationship holds as well. Hence the two optimization problems: selecting V_h's and selecting A_h's, are equivalent. Next we copy costs from the rows given by the $p-1$ remaining tuples to the corresponding remaining $p-1$ rectangles, and modify the costs inside each rectangle in way a similar to the way we modified costs inside the top rectangle. The preprocessing that we have just described allows us to apply the algorithm of Tokuyama and Nakano [26]. Using the notation of the authors, we set λ_d, where d is represented by $d = k_1 + \ldots + k_{r-1} + h$, $1 \leq h \leq k_r$, to our $a_h^r - a_{h-1}^r$, then the weight of the edge (i, j), in the complete bipartite graph with n nodes on the left and $k_1 + \ldots + k_p$ on the right, is set to $t_{j,i}$. Overall, a straightforward implementation of the procedure yields an $O(n^{k_1+\ldots+k_p+3p})$ time algorithm for selecting rows i_h^r and subsets of columns V_h^r. This completes the proof. □

Next we show how to rearrange rows and columns inside nested squares, so that low costs are moved to where a hyperbola is and high costs are left outside of the hyperbola. Given nested submatrices $(V_1 \times V_1) \subset (V_2 \times V_2) \subset \ldots \subset (V_k \times V_k)$ of the cost matrix, one can produce a new cost matrix by rearranging rows and columns inside the submatrices, so that the cost of the new matrix inside the hyperbola is proportionally reduced compared to the costs inside the submatrices. Using the triangle inequality and the symmetricity of the cost matrix, we can bound the sum of costs inside each submatrix $V_h \times V_h$ by the sum of costs at the intersection of row i_h and columns V_h multiplied by twice the cardinality of V_h. This will let us relate the costs after rearrangement to the lower bound on cost established in Lemma 2.

Lemma 3. *Given a hyperbola*

$$H = \left(\bigcup_{h=1}^{k} [a_h] \times [m_h] \right) \cup \left(\bigcup_{h=1}^{k} [m_h] \times [a_h] \right) \ ,$$

k rows i_h, k nested subsets of columns $V_h = [m_h]$, $1 \le h \le k$, and a symmetric cost matrix C that satisfies the triangle inequality, there is a matrix F obtained from C after rearranging rows V_k and columns V_k, so that the sum of costs

$$\sum_{(y,x) \ \in \ H} F_{y,x}$$

of the of rearranged matrix inside the hyperbola is at most

$$4 \sum_{h=1}^{k} a_h \sum_{d \in V_h} C_{i_h,d} \ .$$

The matrix F can be found in $O(n^3)$ time.

Proof. Note that the submatrices are nested $(V_1 \times V_1) \subset (V_2 \times V_2) \subset \ldots \subset (V_k \times V_k)$, and that symmetricity and the triangle inequality implies that the sum of costs of C inside the submatrix $V_h \times V_h$ is at most $2m_h \sum_{j \in V_h} C_{i_h,j}$. Hence a trivial upper bound on the cost of the hyperbola is $4 \sum_{h=1}^{k} m_h \sum_{d \in V_h} C_{i_h,d}$. Our goal is to show how to rearrange rows and columns so as to turn m_h's into a_h's. Let us introduce orientation in matrices: rows are numbered from top to bottom, and columns are numbered from left to right.

We initialize $F_{y,x} := C_{y,x}$ and then rearrange F in two phases. We first rearrange rows to bound the cost inside a *horizontal telescope* defined as $\bigcup_{h=1}^{k} [a_h] \times [m_h]$, and then rearrange columns to bound the cost inside a *vertical telescope* (see Figure 4) defined as $\bigcup_{h=1}^{k} [m_h] \times [a_h]$, while maintaining the previously established bound on the cost of the horizontal telescope.

In the first phase, we focus on the horizontal telescope. We rearrange rows of F in k rounds; first rows V_1, then rows V_2, and so on up to rows V_k. Let σ_s be the sum of costs of F inside the part $([a_1] \times [m_1]) \cup \ldots \cup ([a_s] \times [m_s])$ of the telescope at the end of round s (so σ_k is the sum inside the entire horizontal telescope). Our rearrangements will maintain an invariant that at the end of round s:

(i) the sum of costs inside the part $([a_1] \times [m_1]) \cup \ldots \cup ([a_s] \times [m_s])$ of the telescope is bounded as desired i.e., $\sigma_s \le 2 \sum_{1 \le h \le s} a_h \sum_{j \in V_h} C_{i_h,j}$, and

(ii) for all subsequent submatrices, the initial upper bounds on the sum of costs are maintained i.e., for all $s+1 \le h \le k$, the sum of costs inside the submatrix $V_h \times V_h$ is at most $2m_h \sum_{j \in V_h} C_{i_h,j}$.

The invariant holds for $s = 0$, because of symmetricity, the triangle inequality and the way F was initialized.

During round s, we pick a_s smallest sum rows of the submatrix $V_s \times V_s$, and, if any is below the a_s top rows, we swap the row with the currently highest cost row among the top a_s rows of the submatrix (naturally, we actually swap rows of the entire n by n matrix F inside which the k submatrices reside, because we are not allowed to exchange parts of rows of the matrix). As a result of the swapping, the top a_s rows of the submatrix $V_s \times V_s$ contain rows which sum is the smallest across all possible selections of a_s rows of the submatrix. Hence the sum of costs

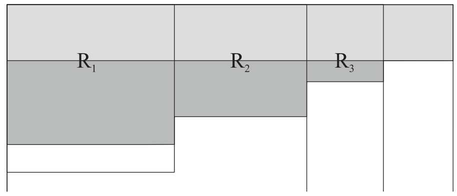


Fig. 5. Three regions inside a horizontal telescope.

inside these rows of the submatrix is at most $2a_s \sum_{j\in V_s} C_{i_s,j}$. Note that since submatrices are nested, swapping preserves upper bounds on the sum of costs for each of the subsequent to $V_s \times V_s$ submatrices i.e., after the swapping, for any $h \geq s+1$, the sum of costs inside the submatrix $V_h \times V_h$ is still at most $2m_h \sum_{j\in V_h} C_{i_h,j}$ (we merely permuted the rows of the submatrix). Now let us bound σ_s from above. By the invariant, the value of σ_{s-1} right before round s was bounded from above by $2\sum_{1\leq h\leq s-1} a_h \sum_{j\in V_h} C_{i_h,j}$. Recall that σ_{s-1} is the sum of costs inside the part $([a_1]\times[m_1]) \cup \ldots \cup ([a_{s-1}]\times[m_{s-1}])$ of the horizontal telescope. We can partition the part into $s-1$ regions: $[a_h]\times[m_{h-1}+1, m_h]$, for $1 \leq h \leq s-1$ (see Figure 5). We notice that after the swapping of rows, the fragments of the $s-1$ regions starting from row a_s+1 downwards (the darkest gray area in Figure 5) contain only some costs that were present in the $s-1$ regions before the swapping. So the sum of costs inside these fragments is bounded from above be the value of σ_{s-1} at the time right before the swapping. We have observed that after the swapping, the cost of the top a_s rows and m_s columns of submatrix $V_s \times V_s$ is at most $2a_s \sum_{j\in V_s} C_{i_s,j}$. Hence, after the swapping, σ_s is bounded from above by $2\sum_{1\leq h\leq s} a_h \sum_{j\in V_h} C_{i_h,j}$.

After k rounds, the invariant implies that the sum of costs of F inside the horizontal telescope is bounded from above by $2\sum_{1\leq h\leq k} a_h \sum_{j\in V_h} C_{i_h,j}$. This conclusion completes the first phase.

In the second phase we rearrange columns. We apply an algorithm that is symmetric to the one used in the first phase (i.e., now we swap columns instead of rows). As a result, we obtain the same upper bound on the sum of costs inside the vertical telescope. The only remaining issue is that we may invalidate the bound on the sum of costs inside the horizontal telescope. This, however, is not the case as explained next.

We will study how locations of costs change and will use labels to track the costs. Each of columns V_k of the matrix F will have a label from the set $\{H, V1, V2, \ldots, Vk\}$ of $k+1$ labels. A label "travels" with a column whenever the column is swapped. We will assign the labels to columns after the first round, and then may modify labels after each round. At the end of any round, we can partition columns into $k+1$ subsets, based on the labels. We will argue that the sum of costs inside columns, contributed by each subset, is appropriately bounded.

After the first round of swapping, we assign label $V1$ to each of the leftmost a_1 columns of F, and each of the columns $[a_1+1, m_k]$ gets the label H. Recall that right before the second phase, the cost inside the horizontal telescope was at most $2\sum_{1\leq h\leq s} a_h \sum_{j\in V_h} C_{i_h,j}$. The swapping of columns done during the first round of the second phase permutes only the columns $[m_1]$, so the cost contributed by columns with label H remains at most $2\sum_{1\leq h\leq s} a_h \sum_{j\in V_h} C_{i_h,j}$. The cost of columns with label $V1$ inside vertical rectangle $[m_1]\times[a_1]$ is at most $2a_1 \sum_{j\in V_1} C_{i_1,j}$.

We will argue that the following invariant holds at the end of any round s

(i) the sum of costs of columns labeled H inside the horizontal telescope is at most the initial bound on the costs inside the horizontal telescope i.e.,

$$\sum_{(y,x)\in H\cap([a_1]\times[m_k]) \ \wedge \ (x \text{ has label } H)} F_{y,x} \leq 2 \sum_{1\leq h\leq k} a_h \sum_{j\in V_h} C_{i_h,j}$$

(ii) for all $1 \leq r \leq s$, the sum of costs of columns labeled Vr inside the part of the hyperbola on and above row m_s

$$\sum_{(y,x)\in H\cap([m_s]\times[m_k]) \ \wedge \ (x \text{ has label } Vr)} F_{y,x}$$

is at most $2a_r \sum_{j\in V_r} C_{i_r,j}$.

The invariant holds at the end of the first round of the second phase. Now we explain how we modify labels during each successive round. At the end of round s, some columns reside among the leftmost a_s columns. These columns get assigned the label Vs. The labels of other columns remain unchanged.

Let us now bound the costs contributed by columns with different labels. The only columns with label Vs are the leftmost a_s columns. Hence, by the argument given for swapping rows, the sum of costs of these columns inside the top m_s rows of the hyperbola is bounded by

$$\sum_{(y,x)\in H\cap([m_s]\times[m_k]) \ \wedge \ (x \text{ has label } Vs)} F_{y,x} \leq 2a_s \sum_{j\in V_s} C_{i_s,j} \ ,$$

as desired. Inside the remaining $m_k - a_s$ columns some new columns might have been swapped in, compared to the columns that were there during the previous round. These new columns can only have label $V(s-1)$, because no other columns move right in round s. So the sum of costs of columns with label Ve, for any $e \leq s-2$, inside the top m_s rows of the hyperbola can only get reduced (as no extra column gets assigned such a label). How about the sum for columns with label $V(s-1)$? We notice that when a column with such label moves right, it is placed at a column where the thickness of the horizontal telescope is at most a_{s-1}. Hence the contribution of this column to the sum of costs of columns with label $V(s-1)$ inside the telescope may only get reduced. Hence the invariant holds after the round s.

After the two phases have been completed, the invariant implies that the sum of costs inside the entire hyperbola is at most

$$4 \sum_{1\leq h\leq k} a_h \sum_{j\in V_h} C_{i_h,j} \ ,$$

which completes the proof. □

Theorem 6. *There is a c-approximation algorithm for the Quorum Deployment problem with a block diagonal hyperbolic quorum matrix and symmetric cost matrix that satisfies the triangle inequality, where $c = 4 \cdot \max_{1\leq r\leq p} k_r$. The algorithm runs in $O(n^{k_1+\ldots+k_p+3p})$ time.*

Proof. We apply Lemma 2 to find sets V_h^r and rows that bound the cost of minimal deployment from below. Then we rearrange rows $V_{k_r}^r$ and columns $V_{k_r}^r$ inside each hyperbola H_r as shown in Lemma 3, and as a result obtain a deployment with a desirable cost. □