

Replica Placement on Bounded Treewidth Graphs

Anshul Aggarwal¹, Venkatesan T. Chakaravarthy², Neelima Gupta¹, Yogish Sabharwal², Sachin Sharma¹, and Sonika Thakral^{1*}

¹ University of Delhi, India `ngupta@cs.du.ac.in`, `sonika.ta@gmail.com`
² IBM Research, India `{vechakra,ysabharwal}@in.ibm.com`

Abstract. We consider the replica placement problem: given a graph and a set of clients, place replicas on a minimum set of nodes of the graph to serve all the clients; each client is associated with a request and maximum distance that it can travel to get served; there is a maximum limit (capacity) on the amount of request a replica can serve. The problem falls under the general framework of capacitated set cover. It admits an $O(\log n)$ -approximation and it is NP-hard to approximate within a factor of $o(\log n)$. We study the problem in terms of the treewidth t of the graph and present an $O(t)$ -approximation algorithm.

1 Introduction

We study a form of capacitated set cover problem [5] called *replica placement* (RP) that finds applications in settings such as data distribution by internet service providers (ISPs) and video on demand service delivery (e.g., [6, 8]). In this problem, we are given a graph representing a network of servers and a set of clients. The clients are connected to the network by attaching each client to a specific server. The clients need access to a database. We wish to serve the clients by placing replicas (copies) of the database on a selected set of servers and clients. While the selected clients get served by the dedicated replicas (i.e., cached copies) placed on themselves, we serve the other clients by assigning them to the replicas on the servers. The assignments must be done taking into account Quality of Service (QoS) and capacity constraints. The QoS constraint stipulates a maximum distance between each client and the replica serving it. The clients may have different demands (the volume of database requests they make) and the capacity constraint specifies the maximum demand that a replica can handle. The objective is to minimize the number of replicas opened. The problem can be formally defined as follows.

Problem Definition (RP): The input consists of a graph $G = (\mathcal{V}, E)$, a set of clients \mathcal{A} and a capacity W . Each client a is attached to a node $u \in \mathcal{V}$, denoted $\text{att}(a)$. For each client $a \in \mathcal{A}$, the input specifies a request $r(a)$ and a distance $d_{\max}(a)$. For a client $a \in \mathcal{A}$ and a node $u \in \mathcal{V}$, let $d(a, u)$ denote the length of the shortest path between u and $\text{att}(a)$, the node to which a is attached

* Corresponding author

- the length is measured by the number of edges and we take $d(a, u) = 0$, if $u = \text{att}(a)$. We say that a client $a \in \mathcal{A}$ can *access* a node $u \in \mathcal{V}$, if $d(a, u)$ is at most $d_{\max}(a)$. A feasible solution consists of two parts: (i) it identifies a subset of nodes $S \subseteq \mathcal{V}$ where a replica is placed at each node in S ; (ii) for each client $a \in \mathcal{A}$, it either opens a dedicated replica at a itself for serving the client's request or assigns the request to the replica at some node $u \in S$ accessible to a . The solution must satisfy the constraint that for each node $u \in S$, the sum of requests assigned to the replica at u does not exceed W . The cost of the solution is the number of replicas opened, i.e., cardinality of S plus the number of dedicated replicas opened at the clients. The goal is to compute a solution of minimum cost. In order to ensure feasibility, without loss of generality, we assume $r(a) \leq W, \forall a \in \mathcal{A}$. \square

The RP problem falls under the framework of the capacitated set cover problem, the generalization of the classical set cover problem wherein each set is associated with a capacity specifying the number of elements it can cover. The latter problem is known to have an $O(\log n)$ -approximation algorithm [5]. Using the above result, we can derive an $O(\log n)$ -approximation algorithm for the RP problem as well. On the other hand, we can easily reduce the classical dominating set problem to RP: given a graph representing an instance of the dominating set problem, we create a new client for each vertex and attach it to the vertex; then, we set $d_{\max}(\cdot) = 1$ for all the clients and $W = \infty$. Since it is NP-hard to approximate the dominating set problem within a factor of $o(\log n)$ [7], by the above reduction, we get the same hardness result for the RP problem as well.

The RP problem is NP-hard even on the highly restricted special case where the graph is simply a path, as can be seen via the following reduction from the bin packing problem. Given K bins of capacity W and a set of items of sizes s_1, s_2, \dots, s_n , for each item i , we create a client a with demand $r(a) = s_i$. We then construct a path of nodes of length K and attach all the clients to one end of the path and take W to be the capacity of the nodes.

Prior Results: Prior work has studied a variant of the RP problem where the network is a directed acyclic graph (DAG), and a client a can access a node u only if there is a directed path from a to u of the length at most $d_{\max}(a)$. Under this setting, Benoit et al. [3] considered the special case of rooted trees and presented a greedy algorithm with an approximation ratio of $O(\Delta)$, where Δ is the maximum degree of the tree. For the same problem, Arora et al. [2] (overlapping set of authors) devised a constant factor approximation algorithm via LP rounding.

Progress has been made on generalizing the above result to the case of bounded treewidth DAGs. Recall that treewidth [4] is a classical parameter used for measuring how close a given graph is to being a tree. For a DAG, the treewidth refers to the treewidth t of the underlying undirected graph. Notice that the reduction from the bin-packing problem shows that the problem is NP-hard even for trees (i.e., $t = 1$) and rules out the possibility of designing an exact algorithm running in time $n^{O(t)}$ (say via dynamic programming) or FPT algorithms with parameter t .

Arora et al. [1] made progress towards handling DAGs of bounded treewidth and designed an algorithm for the case of bounded-degree, bounded-treewidth graphs. Their algorithm achieves an approximation ratio of $O(\Delta + t)$, where Δ is the maximum degree and t is the treewidth of the DAG. Their result also extends for networks comprising of bounded-degree bounded-treewidth subgraphs connected in a tree like fashion.

Our Result and Discussion: We study the RP problem on undirected graphs of bounded treewidth. Our main result is an $O(t)$ -approximation algorithm running in polynomial time (the polynomial is independent of t and the approximation guarantee). In contrast to prior work, the approximation ratio depends only on the treewidth and is independent of parameters such as the maximum degree.

Our algorithm is based on rounding solutions to a natural LP formulation, as in the case of prior work [2, 1]. However, the prior algorithms exploit the acyclic nature of the graphs and the bounded degree assumption to transform a given LP solution to a solution wherein each client is assigned to at most two replicas. In other words, they reduce the problem to a capacitated vertex cover setting, for which constant factor rounding algorithms are known [10].

The above reduction does not extend to the case of general bounded treewidth graphs. Our algorithm is based on an entirely different approach. We introduce the notion of “clustered solutions”, wherein the partially open nodes are grouped into clusters and each client gets served only within a cluster. We show how to transform a given LP solution to a new solution in which a partially-open node participates in at most $(t + 1)$ clusters. This allows us to derive an overall approximation ratio $O(t)$. The notion of clustered solutions may be applicable in other capacitated set cover settings as well.

Other Related Work: The RP problem falls under the framework of the capacitated set cover problem (CSC), which admits an $O(\log n)$ -approximation [5]. Two versions of the CSC problem have been studied: soft capacity and hard capacity settings. Our work falls under the more challenging hard capacity setting, wherein a set can be picked at most once. The capacitated versions of the vertex cover problem (e.g., [10]) and dominating set problem (e.g., [9]) have also been studied. Our result applies to the capacitated dominating problem with uniform capacities and yields $O(t)$ -approximation.

Full Version: Due to space constraints, some of the proofs and details of analysis could not be included in this version. A full version of the paper is available as an Arxiv preprint (<https://arxiv.org/abs/1705.00145>).

2 Overview of the Algorithm

Our $O(t)$ -approximation algorithm is based on rounding solution to a natural LP formulation. In this section, we present an outline of the algorithm highlighting its main features, deferring a detailed description to subsequent sections. We assume that the input includes a decomposition \mathcal{T} of treewidth t of the input network $G = (\mathcal{V}, E)$.

LP Formulation: For each node $u \in \mathcal{V}$, we introduce a variable $y(u)$ to represent the extent to which a replica is opened at u and similarly, for each client $a \in \mathcal{A}$, we add a variable $y(a)$ to represent the extent to which a dedicated replica is opened at a itself. For each client $a \in \mathcal{A}$ and each node $u \in \mathcal{V}$ accessible to a , we use a variable $x(a, u)$ to represent the extent to which a is assigned to u . For a client $a \in \mathcal{A}$ and a node $u \in \mathcal{V}$, we use the shorthand “ $a \sim u$ ” to mean that a can access u .

$$\min \quad \sum_{a \in \mathcal{A}} y(a) + \sum_{u \in \mathcal{V}} y(u)$$

$$y(a) + \sum_{u \in \mathcal{V} : a \sim u} x(a, u) \geq 1 \quad \text{for all } a \in \mathcal{A} \quad (1)$$

$$\sum_{a \in \mathcal{A} : a \sim u} x(a, u) \cdot r(a) \leq y(u) \cdot W \quad \text{for all } u \in \mathcal{V} \quad (2)$$

$$x(a, u) \leq y(u) \quad \text{for all } a \in \mathcal{A} \text{ and } u \in \mathcal{V} \text{ with } a \sim u \quad (3)$$

$$0 \leq y(u), y(a) \leq 1 \quad \text{for all } u \in \mathcal{V} \text{ and } a \in \mathcal{A} \quad (4)$$

Constraint (3) stipulates that a client a cannot be serviced at a node u for an amount exceeding the extent to which u is open. For an LP solution $\sigma = \langle x, y \rangle$, let $\text{cost}(\sigma)$ denote the objective value of σ .

The following simple notations will be useful in our discussion. With respect to an LP solution σ , we classify the nodes into three categories based on the extent to which they are open. A node u is said to be *fully-open*, if $y(u) = 1$; *partially-open*, if $0 < y(u) < 1$; and *fully-closed*, if $y(u) = 0$. A client a is said to be *assigned* to a node u , if $x(a, u) > 0$. For a set of nodes U , let $y(U)$ denote the extent to which the vertices in U are open, i.e., $y(U) = \sum_{u \in U} y(u)$.

Outline: The major part of the rounding procedure involves transforming a given LP solution $\sigma_{\text{in}} = \langle x_{\text{in}}, y_{\text{in}} \rangle$ into an *integrally open solution*: wherein which each node $u \in \mathcal{V}$ is either fully open or closed. Such a solution differs from an integral solution as a client may be assigned to multiple nodes (possibly to its own dedicated replica as well). We address the issue easily via a cycle cancellation procedure to get an integral solution.

The procedure for obtaining an integrally open solution works in two stages. First it transforms the input solution into a “clustered” solution, which is then transformed into an integrally open solution. The notion of clustered solution lies at the heart of the rounding algorithm. Intuitively, in a clustered solution, the set of partially open (and closed) nodes are partitioned into a collection of clusters \mathcal{C} and the clients can be partitioned into a set of corresponding groups satisfying three useful properties, as discussed below.

Let $\sigma = \langle x, y \rangle$ be an LP solution. It will be convenient to express the three properties using the notion of linkage: we say that a node u is *linked* to a node v , if there exists a client a assigned to both u and v . For constants α and ℓ , the solution σ is said to be (α, ℓ) -*clustered*, if the set of partially-open nodes can be partitioned into a collection of clusters, $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ (for some k), such that the the following properties are true:

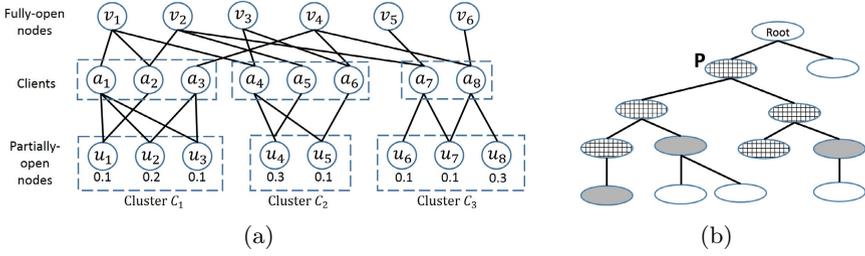


Fig. 1. (a) Illustration for clustered solution. Three clusters are shown C_1, C_2 and C_3 , open to an extent of 0.4, 0.4 and 0.5; the clusters are linked to the sets of fully-open nodes $\{v_1, v_2, v_4\}$, $\{v_1, v_2, v_3, v_4\}$, and $\{v_2, v_4, v_5, v_6\}$. The solution is $(0.5, 4)$ -clustered. (b) Illustration for regions. The figure shows an example tree decomposition. The bags filled solidly represent already identified boundary bags. All checkered bags belong to the region headed by P .

- *Localization*: assignments from clients to the partially-open nodes is localized, i.e., two partially-open nodes are linked only if they belong to the same cluster.
- *Distributivity*: assignments from the clients to fully-open nodes are restricted, i.e., for any C_j , there are at most ℓ fully-open nodes that are linked to the nodes in C_j .
- *Bounded opening*: clusters are tiny, i.e., the total extent to which any cluster is open is at most α , i.e., $y(C_j) < \alpha$.

Figure 1 (a) provides an illustration. In the first stage of the rounding algorithm, we transform the input solution σ_{in} into an $(\alpha, t + 1)$ -clustered solution with the additional guarantee that the number of clusters is at most a constant factor of $\text{cost}(\sigma_{\text{in}})$, where $\alpha \in [0, 1/2]$ is a tunable parameter. The lemma below specifies the transformation performed by the first stage.

Lemma 1. *Fix any constant $\alpha \leq 1/2$. Any LP solution σ can be transformed into a $(\alpha, t + 1)$ -clustered solution σ' such that $\text{cost}(\sigma')$ is at most $2 + 6(t + 1)\text{cost}(\sigma)/\alpha$. Furthermore, the number of clusters is at most $3 + 8 \cdot \text{cost}(\sigma)/\alpha$.*

At a high level, the lemma is proved by considering the tree decomposition \mathcal{T} of the input graph $G = (V, E)$ and performing a bottom-up traversal that identifies a suitable set of boundary bags. We use these boundary bags to split the tree into a set of disjoint regions and create one cluster per region. We then fully open the nodes in the boundary bags and transfer assignments from the nodes that stay partially-open to these fully-open nodes. The transfer of assignments is performed in such a manner that clusters get localized and have distributivity of $(t+1)$. By carefully selecting the boundary bags, we shall enforce that each cluster is open to an extent of only α and that the number of clusters is also bounded. The proof is discussed in Section 3.

The goal of the second stage is to transform a $(1/4, t + 1)$ -clustered solution (obtained from Lemma 1) into an integrally open solution. At a high level, the

localization property allows us to independently process each cluster $C \in \mathcal{C}$ and its corresponding group of clients A . The clients in A are assigned to a set of fully-open nodes, say F . For each node $u \in F$, we identify a suitable node $v \in C$ called the “consort” of $u \in C$ and fully open v . Then the idea is to transfer assignments from the non-consort nodes to the nodes in F and their consorts in such a manner that at the end, no client is assigned to the non-consort nodes. This allows us to fully close the non-consort nodes. The localization and bounded opening properties facilitate the above maneuver. On the other hand, the distributivity property ensures that F is at most $(t + 1)$. This means that we fully open at most $(t + 1)$ consorts per cluster. Thus, overall increase in cost is at most $(t + 1)|\mathcal{C}|$. Since $|\mathcal{C}|$ is guaranteed to be linear in $\text{cost}(\sigma_{\text{in}})$, we get an $O(t)$ approximation factor.

Lemma 2. *Let $\sigma = \langle x, y \rangle$ be a $(1/4, t + 1)$ -clustered solution via a collection of clusters \mathcal{C} . The solution can be transformed into an integrally open solution $\sigma' = \langle x', y' \rangle$ such that $\text{cost}(\sigma') \leq 2 \cdot \text{cost}(\sigma) + 2(t + 1)|\mathcal{C}|$.*

Once we obtain an integrally open solution, it can be transformed to an integral solution by applying a cycle cancellation strategy, as given by the following lemma.

Lemma 3. *Any integrally open solution $\sigma = \langle x, y \rangle$ can be transformed to an integral solution $\sigma' = \langle x', y' \rangle$ such that $\text{cost}(\sigma') \leq 4 \cdot \text{cost}(\sigma)$.*

We can transform any input LP solution σ_{in} into an integral solution σ_{out} by applying the above three transformations leading to the following main result of the paper: the RP problem admits on $O(t)$ -approximation poly-time algorithm.

3 Clustered Solutions: Proof of Lemma 1

The goal is to transform a given solution into an $(\alpha, t + 1)$ -clustered solution with the properties claimed in the lemma. The idea is to select a set of partially-open or closed nodes and open them fully, and then transfer assignments from the other partially-open nodes to them in such a manner that the partially-open nodes get partitioned into clusters satisfying the three properties of clustered solutions. An issue in executing the above plan is that the capacity at a newly opened node may be exceeded during the transfer. We circumvent the issue by first performing a pre-processing step called de-capacitation.

3.1 De-capacitation

Consider an LP solution $\sigma = \langle x, y \rangle$ and let u be a partially-open or closed node. The clients that can access u might have been assigned to other partially-open nodes under σ . We call the node u *de-capacitated*, if even when all the above assignments are transferred to u , the capacity at u is not exceeded; meaning,

$$\sum_{a \sim u} \sum_{v: a \sim v \wedge v \in \text{PO}} x(a, v) < W,$$

For each partially-open node v (considered in an arbitrary order)
 For each client a that can access both u and v (considered in an arbitrary order)
 Compute capacity available at u : $\text{cap}(u) = W - \sum_{b \in \mathcal{A} : b \sim u} x(b, u) \cdot r(b)$
 If $\text{cap}(u) = 0$ exit the procedure
 $\delta = \min \left\{ x(a, v), \frac{\text{cap}(u)}{r(a)} \right\}$
 Increment $x(a, u)$ by δ and decrement $x(a, v)$ by δ .

Fig. 2. Pulling procedure for a given partially-open or closed node u .

where PO is the set of partially-open nodes under σ (including u). The solution σ is said to be *de-capacitated*, if all the partially-open and the closed nodes are de-capacitated.

The preprocessing step transforms the input solution into a de-capacitated solution by performing a pulling procedure on the partially-open and closed nodes. Given a partially-open or closed node u , the procedure transfers assignments from other partially-open nodes to u , as long as the capacity at u is not violated. The procedure is shown in Figure 2, which we make use of in other components of the algorithm as well.

Lemma 4. *Any LP solution $\sigma = \langle x, y \rangle$ can be transformed into a de-capacitated solution $\sigma' = \langle x', y' \rangle$ such that $\text{cost}(\sigma') \leq 2 \cdot \text{cost}(\sigma)$.*

Proof. We consider the partially-open and closed nodes, and process them in an arbitrary order, as follows. Let u be a partially-open or closed node. Hypothetically, consider applying the pulling procedure on u . The procedure may terminate in one of two ways: (i) it exits mid-way because of reaching the capacity limit; (ii) the process executes in its entirety. In the former case, we fully open u and perform the pulling procedure on u . In the latter case, the node u is de-capacitated and so, we leave it as partially-open or closed, without performing the pulling procedure. It is clear that the above method produces a de-capacitated solution σ' . We next analyze the cost of σ' . Let s be the number of partially-open or closed nodes converted to be fully-open. Apart from these conversions, the method does not alter the cost and so, $\text{cost}(\sigma')$ is at most $s + \text{cost}(\sigma)$. Let the total amount of requests be $r_{\text{tot}} = \sum_{a \in \mathcal{A}} r(a)$. The extra cost s is at most $\lceil r_{\text{tot}}/W \rceil$, since any newly opened node is filled to its capacity. Due to the capacity constraints, the input solution σ must also incur a cost of at least $\lceil r_{\text{tot}}/W \rceil$. It follows that $\text{cost}(\sigma')$ is at most $2 \cdot \text{cost}(\sigma)$. \square

3.2 Clustering

Given Lemma 4, assume that we have a de-capacitated solution $\sigma = \langle x, y \rangle$. We next discuss how to transform σ into an $(\alpha, t + 1)$ -clustered solution. The transformation would perform a bottom-up traversal of the tree decomposition and identify a set of partially-open or closed nodes. It would then fully open

them and perform the pulling procedure on these nodes. The advantage is that the above nodes are de-capacitated and so, the pulling procedure would run to its entirety (without having to exit mid-way because of reaching capacity limits). As a consequence, the linkage between the nodes gets restricted, leading to a clustered solution. Below we first describe the transformation and then present an analysis.

Transformation: Consider the given tree decomposition \mathcal{T} . We select an arbitrary bag of \mathcal{T} and make it the root. A bag P is said to be an *ancestor* of a bag Q , if P lies on the path connecting Q and the root; in this case, Q is called a *descendant* of P . We consider P to be both an ancestor and descendant of itself. A node u may occur in multiple bags; among these bags the one closest to the root is called the *anchor* of u and it is denoted $\text{anchor}(u)$. A *region* in \mathcal{T} refers to any set of contiguous bags (i.e., the set of bags induce a connected sub-tree).

In transforming σ into a clustered solution, we shall encounter three types of nodes and it will be convenient to color them as red, blue and brown. To start with, all the fully-open nodes are colored red and the remaining nodes (partially-open nodes and closed nodes) are colored blue. The idea is to carefully select a set of blue nodes, fully-open them and perform the pulling procedure on these nodes; these nodes are then colored brown. Thus, while the blue nodes are partially-open or closed, the red and the brown nodes are fully-open, with the brown and blue nodes being de-capacitated.

The transformation identifies two kinds of nodes to be colored brown: *helpers* and *boundary nodes*. We say that a red node $u \in \mathcal{V}$ is *proper*, if it has at least one neighbor $v \in \mathcal{V}$ which is a blue node. For each such proper red node u , we arbitrarily select one such blue neighbor $v \in \mathcal{V}$ and declare it to be the helper of u . Multiple red nodes are allowed to share the same helper. Once the helpers have been identified, we color them all brown. The boundary brown nodes are selected via a more involved bottom-up traversal of \mathcal{T} that works by identifying a set \mathcal{B} of bags, called the *boundary bags*. To start with, \mathcal{B} is initialized to be the empty set. We arrange the bags in \mathcal{T} in any bottom-up order (i.e., a bag gets listed only after all its children are listed) and then iteratively process each bag P as per the above order. Consider a bag P . We define the *region headed by* P , denoted $\text{Region}(P)$, to be the set of bags Q such that Q is a descendant of P , but not the descendant of any bag already in \mathcal{B} . See Figure 1 (b) for an illustration. A blue node u is said to be *active at* P , if it occurs in some bag included in $\text{Region}(P)$. Let $\text{active}(P)$ denote the set of blue nodes active at P . We declare P to be a boundary bag and add it to \mathcal{B} under three scenarios: (i) P is the root bag. (ii) P is the anchor of some red node. (iii) the extent to which the nodes in $\text{active}(P)$ are open is at least α , i.e., $\sum_{u \in \text{active}(P)} y(u) \geq \alpha$. If P is identified as a boundary bag, then we select all the blue nodes appearing in the bag and change their color to be brown. Once the bottom-up traversal is completed, we have a set of brown nodes (helpers and boundary nodes). We consider these nodes in any arbitrary order, open them fully, and perform the pulling procedure on them. We take σ' to be the solution obtained by the above process. This completes the construction of σ' . We note that a node may change

its color from blue to brown in the above process, and the new color is to be considered while determining the active sets thereafter. Notice that during the whole process of the above transformation, the solution continues to remain de-capacitated.

Analysis: We now show that σ' is an $(\alpha, t + 1)$ -clustered solution. To start with, we have a set of red nodes that are fully-open and a set of blue nodes that are either partially-open or closed under σ . The red nodes do not change color during the transformation. On the other hand, each blue node u becomes active at some boundary bag P . If u occurs in the bag P , it changes its color to brown, otherwise it stays blue. Thus, the transformation partitions the set of originally blue nodes into a set of brown nodes and a set of nodes that stay blue. In the following discussion, we shall use the term ‘blue’ to refer to the nodes that stay blue. With respect to the solution σ' , the red and brown nodes are fully-open, whereas the blue nodes are partially-open or closed.

Recall that with respect to σ' , two nodes u and v are linked, if there is a client a assigned to both u and v . In order to prove the properties of $(\alpha, t + 1)$ -clustering, we need to analyze the linkage information for the blue nodes. We first show that the blue nodes cannot be linked to brown nodes, by proving the following stronger observation.

Proposition 1. *If a client $a \in \mathcal{A}$ is assigned to a blue node u under σ' , then a cannot access any brown node v .*

Proposition 1 rules out the possibility of a blue node u being linked to any brown node. Thus, u may be linked to a red node or another blue node. The following lemma establishes a crucial property on the connectivity in these two settings.

Lemma 5. *(a) If two blue nodes u and v are linked under σ' , then there must exist a path connecting u and v consisting of only blue nodes. (b) If a blue node u is linked to a red node v under σ' , then there must exist a path p connecting u and v such that barring v , the path consists of only blue nodes.*

The transformation outputs a set of boundary bags \mathcal{B} ; let $\overline{\mathcal{B}}$ denote the set of non-boundary bags. If we treat the bags in \mathcal{B} as cut-vertices and delete them from \mathcal{T} , the tree splits into a collection \mathcal{R} of disjoint regions. Alternatively, these regions can be identified in the following manner. For each bag $P \in \mathcal{B}$ and each of its non-boundary child $Q \in \overline{\mathcal{B}}$, add the region headed by Q ($\text{Region}(Q)$) to the collection \mathcal{R} . Let the collection derived be $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$. It is easy to see that \mathcal{R} partitions $\overline{\mathcal{B}}$ and that the regions in \mathcal{R} are pairwise disconnected (not connected by edges of the tree decomposition).

In order to show that σ' is an $(\alpha, t + 1)$ -clustered solution, let us suitably partition the set of blue nodes into a collection of clusters \mathcal{C} . For each region R_j , let C_j be the set of partially-open nodes that occur in some bag of R_j . We take \mathcal{C} to be the collection $\{C_1, C_2, \dots, C_k\}$. It can be verified that the collection \mathcal{C} is a partitioning of the set of partially-open nodes. Based on Lemma 5 we can establish the following result.

Lemma 6. *The solution σ' is $(\alpha, t + 1)$ -clustered.*

We next analyze the cost of the solution $\sigma' = \langle x', y' \rangle$. Let **Red**, **Blue** and **Brown** denote the set of red, brown and blue nodes, respectively. We can see that in constructing σ' the brown nodes are the only new nodes opened fully and hence, $\text{cost}(\sigma') \leq \text{cost}(\sigma) + |\mathbf{Brown}|$. We create a brown helper node for each red node and furthermore, for each boundary bag $P \in \mathcal{B}$, we convert all the blue nodes in P to be brown and the number of blue nodes per bag is at most $(t + 1)$. Thus, the number of brown nodes is at most $|\mathbf{Red}| + (t + 1)|\mathcal{B}|$. A bag P is made into a boundary bag under one of the three scenarios: (i) P is the root bag; (ii) P is the anchor of some red node; (iii) the total extent to which the nodes in $\text{active}(P)$ are open is at least α . The number of boundary bags of the first two types are $1 + |\mathbf{Red}|$ and those of the third type can be $(1/\alpha)$ times the extent to which the blue nodes are open, which is in turn, at most $\text{cost}(\sigma)$. Using the above arguments, we can show that $|\mathcal{B}|$ is at most $2 + |\mathbf{Red}| + \text{cost}(\sigma)/\alpha$ and $\text{cost}(\sigma')$ is at most $2 + 3(t + 1)\text{cost}(\sigma)/\alpha$. The preprocessing step of de-capacitation incurs a 2-factor increase in cost. Taking this into account, we get the cost bound claimed in the statement of Lemma 1.

As mentioned earlier, an issue with the collection \mathcal{C} is that it may have more clusters than the bound claimed in Lemma 1. The issue can be resolved as follows. Consider each boundary bag P . All the non-boundary children of P have a corresponding cluster in \mathcal{C} and let \mathcal{C}_P denote the collection of these clusters. We merge any two clusters C, C' from \mathcal{C}_P having $y(C), y(C') \leq \alpha/2$. The process is stopped when we cannot find two such clusters.

It can be shown that the process of merging does not affect distributivity and the total number of clusters in the transformed solution is at most $3 + 4\text{cost}(\sigma)/\alpha$. The preprocessing step of de-capacitation incurs a 2-factor increase in cost. Taking this into account, we get the bound on number of clusters claimed in the statement of Lemma 1.

4 Integrally Open Solution: Proof of Lemma 2

Our goal is to transform a given $(1/4, t + 1)$ -clustered solution $\sigma = \langle x, y \rangle$ into an integrally open solution σ' . We classify the clients into two groups, *small* and *large*, based on the extent to which they are served by dedicated replicas: a client $a \in \mathcal{A}$ said to be *small*, if $y(a) \leq 1/2$, and it is said to be *large* otherwise. Let \mathcal{A}_s and \mathcal{A}_l denote the set of small and large clients, respectively.

We pre-process the solution σ by opening a dedicated replica at each large client a and removing its assignments to the nodes (set $y(a) = 1$ and set $x(a, u) = 0$ for all nodes u accessible to a). We see that the transformation at most doubles the cost and the solution remains $(1/4, t + 1)$ -clustered.

Consider the pre-processed solution σ . Let \mathcal{C} denote the set of clusters (of the partially-open nodes) under σ . For each cluster $C \in \mathcal{C}$, we shall fully open a selected set of at most $2(t + 1)$ nodes and fully close rest of the nodes in it.

We now describe the processing for a cluster $C \in \mathcal{C}$. Let $A \subseteq \mathcal{A}_s$ denote the set of clients assigned to the nodes in C . By the distributivity property, these clients

are assigned to at most $(t+1)$ fully-open nodes, denoted $F = \{u_1, u_2, \dots, u_{t+1}\}$. A client $a \in A$ may be assigned to multiple nodes from F . In our procedure, it would be convenient if each client is assigned to at most one node from F and we obtain such a structure using the following transformation.

Proposition 2. *Given a solution $\sigma = \langle x, y \rangle$, a set of fully-open nodes F and a set of clients A , we can obtain a solution $\sigma' = \langle x', y' \rangle$ such that each client $a \in A$ is assigned to at most one node from F . Furthermore, the transformation does not alter the other assignments, i.e., for any node $u \in \mathcal{V}$ and any client $a \in \mathcal{A}$, if $u \notin F$ or $a \notin A$, then $x'(a, u) = x(a, u)$. Moreover, $\text{cost}(\sigma') \leq \text{cost}(\sigma) + |F|$.*

The above proposition is proved via a cycle cancellation procedure that transfers assignments amongst the nodes in F . The procedure can ensure that, except for at most $|F|$ clients, every other client $a \in \mathcal{A}$ is assigned to at most one node from F . We open dedicated replicas at the exceptional clients and this results in an cost increase of at most $|F|$.

The proposition does not alter the other assignments and so, its output solution is also $(1/4, t+1)$ -clustered. Given the proposition and the pre-processing, we can assume that $\sigma = \langle x, y \rangle$ is $(1/4, t+1)$ -clustered wherein each client $a \in A$ is assigned to at most one node from F and that $y(a) \leq 1/2$. For each node $u_i \in F$, let $A_i \subseteq A$ denote the set of clients assigned to the node u_i . The proposition guarantees that these sets are disjoint.

For a node v and a client a , let $\text{load}(a, v)$ denote the amount of load imposed by a on v towards the capacity: $\text{load}(a, v) = x(a, v)r(a)$. It will be convenient to define the notion over sets of clients and nodes. For a set of clients B and a set of nodes U , let $\text{load}(B, U)$ denote the load imposed by the clients in B on the nodes U : $\text{load}(B, U) = \sum_{a \in B, v \in U: a \sim v} x(a, v)r(a)$; when the sets are singletons, we shall omit the curly braces. Similarly, for a subset $C' \subseteq C$, let $\text{load}(C') = \sum_{v \in C'} \text{load}(v)$.

The intuition behind the remaining transformation is as follows. We shall identify a suitable set of nodes $L = \{v_1, v_2, \dots, v_{t+1}\}$ from C , with v_i being called the *consort* of u_i in C , and fully open all these nodes. Then, we consider the non-consort nodes $C' = C - L$ and for each $i \leq t+1$, we transfer the load $\text{load}(A_i, C')$ to the node u_i . As a result, no clients are assigned to the non-consort nodes any more and so, they can be fully closed. In order to execute the transfer, for each $i \leq t+1$, we create space in u_i by pushing a load equivalent to $\text{load}(A_i, C')$ from u_i to its (fully-opened) consort v_i . The amount of load $\text{load}(A_i, C')$ involved in the transfer is very small: the bounded opening property ensures that $y(C) < 1/4$ and thus, $\text{load}(A_i, C') < W/4$. The fully-opened consort v_i has enough additional space to receive the load: $y(v_i) \leq 1/4$ and so, $\text{load}(A, v_i) \leq W/4$, which means that if we fully open the consort, we get an additional space of $(3/4)W$. However, an important issue is that a consort v_i may not be accessible to all the clients in A_i . Therefore, we need to carefully choose the consorts in such a manner that each fully open node u_i has enough load accessible to the consort v_i that can be pushed to v_i . Towards this purpose, we define the notion of *pushable load*. For a node $u_i \in F$ and a node $v \in C$, let $\text{pushable}(u_i, v)$ denote the amount

of load on u_i that is accessible to v : $\text{pushable}(u_i, v) = \sum_{a \in A_i: a \sim v} x(a, u_i)r(a)$. We next show how to identify a suitable set of consorts such that the pushable load is more than the load that we wish to transfer.

Lemma 7. *We can find a set of nodes $L = \{v_1, v_2, \dots, v_{t+1}\}$ such that for all $i \leq t + 1$, $\text{pushable}(u_i, v) \geq \text{load}(A_i, C')$.*

We have shown that each node u_i has a load of at least $\text{load}(A_i, C')$ which can be pushed to its consort v_i . As observed earlier $\text{load}(A_i, C') < W/4$ and $\text{load}(A_i, v_i) \leq W/4$. Hence, when we fully open the consort, we get an additional space of $(3/4)W$, which is sufficient to receive the load from u_i .

Given the above discussion, we iteratively consider each cluster $C_j \in \mathcal{C}$ and perform the above transformation. This results in $(t + 1)$ consorts from C_j being fully-opened and all the other nodes in C_j being fully closed. At the end of processing all the clusters, we get a solution in which each node either fully open or fully close. For each cluster C_j , we incur an extra cost of at most $(t + 1)$ while applying Proposition 2, and an additional cost of $(t + 1)$ for opening the consorts. Thus, the cost increases by at most $2(t + 1)|\mathcal{C}|$.

References

1. S. Arora, V. Chakaravarthy, K. Gupta, N. Gupta, and Y. Sabharwal. Replica placement on directed acyclic graphs. In V. Raman and S. Suresh, editors, *Proceedings of the 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 213–225, 2014.
2. S. Arora, V. Chakaravarthy, N. Gupta, K. Mukherjee, and Y. Sabharwal. Replica placement via capacitated vertex cover. In A. Seth and N. Vishnoi, editors, *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 263–274, 2013.
3. A. Benoit, H. Larchevêque, and P. Renaud-Goud. Optimal algorithms and approximation algorithms for replica placement with distance constraints in tree networks. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1022–1033, 2012.
4. H. Bodlaender and A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Computer Journal*, 51(3):255–269, 2008.
5. J. Chuzhoy and J. Naor. Covering problems with hard capacities. *SIAM Journal of Computing*, 36(2):498–515, 2006.
6. I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Computer Networks*, 40:205–218, 2002.
7. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.
8. K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12:628–637, 2001.
9. M. Kao, H. Chen, and D. Lee. Capacitated domination: Problem complexity and approximation algorithms. *Algorithmica*, 72(1):1–43, 2015.
10. B. Saha and S. Khuller. Set cover revisited: Hypergraph cover with hard capacities. In A. Czumaj, K. Mehlhorn, A. Pitts, and R. Wattenhofer, editors, *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7391 of LNCS, pages 762–773. Springer, 2012.



<http://www.springer.com/978-3-319-62126-5>

Algorithms and Data Structures

15th International Symposium, WADS 2017, St. John's,

NL, Canada, July 31 - August 2, 2017, Proceedings

Faith, E.; Kolokolova, A.; Sack, J.-R. (Eds.)

2017, XX, 594 p. 129 illus., Softcover

ISBN: 978-3-319-62126-5