A Primal-Dual Resource Augmentation Analysis of a Constant Approximate Algorithm for Stable Coalitions in a Cluster

Nedialko B. Dimitrov*
University of Texas at Austin
1 University Station
Austin, Texas
ned@cs.utexas.edu

Indrajit Roy†
University of Texas at Austin
1 University Station
Austin, Texas
indrajit@cs.utexas.edu

ABSTRACT

In this paper we study the following Cluster Profit Problem. Highly parallelizable requests are present on some network nodes. Each request is associated with a tuple (g, r). The requester is willing to pay kg if k machines execute the request in parallel. If some machines work on a request, the machines must pay the request processing cost, r, as well as connection costs to the request. The problem is to find a profit maximizing assignment of machines to requests, such that each machine works on at most one request. The Cluster Profit Problem can be viewed as a profit maximizing variant of the Facility Location Problem. We provide and analyze an algorithm under resource augmentation for the Cluster Profit Problem. Resource augmentation is a technique made famous by the LRU caching analysis. We compare our algorithm with the optimal algorithm operating on a network graph that has a constant factor longer distances. We prove our algorithm is a constant approximation under this resource augmentation. We also show that our algorithm is resilient to group deviations if deviating increases communication costs by a constant factor.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Economics, Theory

Keywords

Primal-dual, resource augmentation, facility location, approximate core equilibrium

*Supported by NSF Grant CCF-0635203.

†Supported by MCD Fellowship.

SPAA'08, June 14–16, 2008, Munich, Germany.

1. INTRODUCTION

In this paper, we study the following Cluster Profit Problem. Consider a network where some of the nodes can perform computations, we call these nodes machines. Highly parallelizable requests for computation are present on some nodes of the network. Each request is associated with a tuple (g, r). The first parameter signifies that the requester is willing to pay kg if the request is executed in parallel by k machines. If a set $\mathcal{A}$ of machines is assigned to a request, the machines in $\mathcal{A}$ must collectively pay r, the cost required to satisfy the request. The machines in $\mathcal{A}$ must also pay communication costs equal to the sum of the shortest paths from each machine in $\mathcal{A}$ to the request. The Cluster Profit Problem is to assign each machine to at most one request such that the profit to the system is maximized. The Cluster Profit Problem is formally defined in Section 2.

Though we have chosen the cluster setting in specific to describe the Cluster Profit Problem (CPP), it has more general interpretations.

First, the problem can be viewed as a variant of the Facility Location Problem [4]. Suppose, we are given a graph where some nodes are labeled as customers and some as facilities. Each facility is associated with a tuple (g, r). The number r signifies the opening cost of the facility. The number g signifies the quality of the facility, stating that each customer is willing to pay g for being connected to this facility. If we connect a customer to a facility, we are required to pay a shipping cost equal to the distance from the customer to the facility. We are also required to pay a facility's opening cost if we connect any customer to the facility. Finding a profit maximizing assignment of customers to facilities is exactly CPP. In contrast, the standard Facility Location Problem has no quality values. Instead we must assign each customer to a facility and minimize the overall cost of the assignment.

Second, CPP can be viewed as an instance of a coalition formation problem. In a general coalition formation setting, we are given a set of tasks and a set of agents. We would like to partition the agents and assign each resulting subset to some task [18]. Often, we would like to have the partition and subsequent assignment of tasks maximize some utility function [17]. CPP can be viewed as a coalition formation problem, where the tasks are the requests, the agents are the machines, and the utility function is defined by the profit.

Computation in a large scale distributed system motivate some desirable properties in an algorithm for CPP. One such property is that the algorithm should approximately maximize the revenue of the system. Lemma B.1 shows that CPP is NP-hard, thus, unless NP = P, we cannot hope for an efficient algorithm that gives the maximum revenue.

Another property is that the algorithm should be resilient to selfish behavior of the system's participants. If a system is spread across nodes owned by several entities, each entity may have its selfish interests override the algorithm's requirements. Thus, we would like an algorithm that is resilient to such group deviations.

1.1 Resource Augmentation

In this paper we analyze an algorithm for CPP under the notion of resource augmentation(RA) [10, 16].

The technique of resource augmentation has been used to prove results for LRU caching in the competitive algorithms literature [1, 21]. . In resource augmentation the analyzed algorithm competes with a constant-factor resource advantage over the optimal algorithm, OPT.In the case of LRU analysis, the resource advantage is constant-factor larger cache size. In our case, we introduce a resource augmentation through the constant $\theta \geq 1$. Our algorithm operates on an instance where distances are shorter by the constant factor θ.

This particular type of resource augmentation has a natural interpretation. Consider a large company with millions of machines which the company does not own. The company suggests coalitions to the participating machines. If the machines follow the company's suggestion, they may use the company's proprietary bandwidth to communicate. If they deviate, the machines must use some other method of communication and incur slightly larger communication costs. The parameter θ captures this increase in the communication costs.

In this paper we give a constant RA approximate algorithm for the Cluster Profit Problem. We formally define a constant RA approximate algorithm as follows. Consider a maximization problem and an algorithm A_1 for solving the maximization problem. Let OPT be an algorithm achieving an optimal solution to the problem. Let L be a transformation that takes away some resources. We say algorithm A_1 is a γ-RA approximate algorithm under L if for all instances x we have $\mathrm{OPT}(\mathrm{L}(x)) \leq \gamma \mathrm{A}_1(x)$. In our paper, the transformation L is fixed to increasing the distances by the constant factor θ. Since the particular transformation is fixed, from here on we simply use the term γ-RA approximate algorithm.

Paralleling the definition of constant RA approximation, in this paper we study constant RA stability. This concept is formally defined in Section 5. Informally, consider the example introduced in this section. An algorithm suggested by the company is stable if the extra communication costs incurred by any deviating group of machines is more than the profit achieved by deviation. Such an algorithm would be a 1-RA stable algorithm under L.

1.2 Main Contributions

In this paper, we design an algorithm that is constant RA approximate and constant RA stable. Specifically in Section 3, we describe an algorithm for CPP called Algorithm ALG with the following properties:

- We show ALG gives a constant RA approximation. In specific, we show Theorem 1 stating that ALG operating on an instance of CPP gains at least a constant fraction of the profit of an optimal algorithm operating on the same instance, but with edge lengths increased by a constant factor. See Section 4 for further details.
- We show ALG is constant RA stable to group deviations. In specific, we show Theorem 2 stating that every subset of the machines gains at least a constant fraction of the profit they would gain under an optimal deviation, if deviating increases communication costs by a constant factor. See Section 5 for further details.

In addition, in the appendix we explain how to implement ALG in a distributed fashion. In specific, we show that if the set of machines is $\mathcal{U}$ and the set of requests is $\mathcal{V}$, the algorithm terminates in $O(|\mathcal{V}|)$ phases using $O(|\mathcal{U}||\mathcal{V}|^2)$ local messages.

1.3 Related Work

Our paper most closely parallels work on the Facility Location Problem (FLP). FLP has been studied for decades [11, 7, 4, 14]. Of this work, the works closest to our own are a sequence of papers using primal-dual algorithms for approximating the FLP. Jain and Vazirani describe a greedy primal-dual algorithm that gives a 3-approximate solution to the FLP [9]. Mettu and Plaxton give an alternate interpretation of the Jain and Vazirani algorithm that is more suitable for implementing in a distributed manner [13]. Jain et al. give an algorithm based on similar ideas as Jain and Vazirani, but use an alternate analysis technique called a factor-revealing LP to improve the approximation factor [8].

Our algorithm is based on a similar approach as the Jain and Vazirani algorithm, however, it is designed for CPP, which is a fundamentally different problem than FLP. FLP minimizes the cost of an assignment, but CPP maximizes the profit of an assignment. As there can be assignments with high costs that also yield high profits, a facility location algorithm that finds the approximately lowest cost assignment provides no guarantees on maximizing the profit earned. Therefore, algorithms for FLP do not apply directly to our problem.

In analyzing our algorithm, we use a factor-revealing LP analysis. However, we extend the factor-revealing LP technique of Jain et al. by combining it with resource augmentation analysis to show our algorithm is constant RA approximate. In addition, we use an interpretation similar to that of Mettu and Plaxton as a basis for our distributed algorithm.

Recently, game theoretic stability in distributed systems has been studied [5, 3, 12]. Most work concentrates on showing that a particular distributed system is in Nash equilibrium. In a system in Nash equilibrium, we are guaranteed that no individual has incentive to unilaterally deviate. However, Nash equilibrium does not provide guarantees against group deviation.

Group deviations are best modeled with the core equilibrium, a notion from cooperative game theory [20]. Pal and Tardos develop a method of cost sharing in a Facility Location Game, using ideas from the primal-dual facility location approximation algorithms [15]. Furthermore, Pal and Tardos show their cost sharing method is in the approximate core.

Contrasting with the work on Nash equilibrium, we show our algorithm is resilient to group deviations. Our stability results are similar to the results of Pal and Tardos, in that both depend on the underlying primal-dual algorithm. Since the facility location algorithms are approximately optimal, Pal and Tardos show their cost sharing method is in the approximate core. On the other hand, our algorithm is constant RA approximate and we show our profit shares are constant RA stable. Furthermore, our proof technique is different from the proof technique of Pal and Tardos.

Throughout the rest of the paper, due to space considerations, we omit most technical proofs. Instead, we provide the lemma statements, some intuition, and include full proofs in the appendix.

2. THE CLUSTER PROFIT PROBLEM

In this section, we formally define the Cluster Profit Problem (CPP). Let $G = (\mathcal{N}, \mathcal{E})$ be a graph with a non-negative distance metric d. Let a subset of the nodes, $\mathcal{U} \subseteq \mathcal{N}$, denote a set of machines. Let another subset of nodes, $\mathcal{V} \subseteq \mathcal{N}$, denote received requests for computation. The request at node $v \in \mathcal{V}$ is associated with a tuple of non-negative real numbers, (g_v, r_v). The number g_v denotes that the requester is willing to pay g_v for each machine that is assigned to the request. If set $\mathcal{A}$ of machines is assigned to request v, the machines in $\mathcal{A}$ must collectively pay r_v, the cost required to satisfy the request, and $\sum_{u \in \mathcal{A}} d_{u,v}$, where $d_{u,v}$ is the distance of the shortest path from machine u to the request v. The Cluster Profit Problem asks to find a coalition configuration, or assignment of machines to requests, such that each machine is assigned to at most one request and the profit of the system is maximized.

By Lemma B.1, CPP is NP-hard. To provide theoretical guarantees for efficiency and optimality, we introduce two constants. The examples in Section 3.1, give further intuition on how these two parameters help in proving our approximation results. First, we introduce a constant ω such that $r_v \leq \omega g_v$. In essence, this restriction means that the requester pays at least the resource cost, as long as ω machines are working on the request.

As discussed in Section 1.1, we also introduce a constant $\theta \geq 1$. If our algorithm operates on a graph G with distance metric d, we compare with OPT operating on a graph G' with distance metric $\theta \cdot d$. Thus, our algorithm has the resource advantage of distances that are shorter by the constant factor θ.

3. ALGORITHM DESCRIPTION

We use an algorithmic approach inspired by a sequence of papers starting with an approximation algorithm to the set cover problem and including some recent papers on the Facility Location Problem [8, 2, 13, 19]. The key idea behind the algorithm is to greedily find a subset of machines and a request such that the corresponding assignment gives the maximum profit density.

Informally, we use the concept of a *star*, $S = (v_S, \mathcal{A}_S)$, to denote a tuple of a request, v_S, and a set of machines, $\mathcal{A}_S$ [8]. In our presentation, we abuse notation slightly by writing $u \in S$ and $|S|$ when we mean $u \in \mathcal{A}_S$ and $|\mathcal{A}_S|$, respectively. Define the profit of a star $S = (v_S, \mathcal{A}_S)$ as $p_S = (|S| \cdot g_{v_S} - \sum_{u \in S} d_{u,v_S} - r_{v_S})$. Correspondingly, the profit density of the star S is $p_S/|S|$. Our greedy algorithm finds the star $S^* = (v_{S^*}, \mathcal{A}_{S^*})$ with the highest profit density and assigns the machines in $\mathcal{A}_{S^*}$ to work on the request v_{S^*}. The algorithm then sets the resource cost $r_{v_{S^*}}$ to zero, removes the machines in $\mathcal{A}_{S^*}$ from the set of available machines, and iterates to find the next best star. The algorithm terminates when there are no unassigned machines left or the profit density of all remaining stars is negative.

$$\begin{aligned} \max \quad & \sum_{S \in \mathcal{T}} p_S \cdot x_S && && \text{(P)} \\ \text{s.t.} \quad & && && \\ & \sum_{S \in \mathcal{T} : u \in S} x_S \leq 1 && \forall u \in \mathcal{U} && (1) \\ & x_S \in \{0,1\} && \forall S \in \mathcal{T} && (2) \end{aligned}$$

(a) Primal

$$\begin{aligned} \min \quad & \sum_{u \in \mathcal{U}} \alpha_u && && \text{(D)} \\ \text{s.t.} \quad & && && \\ & \sum_{u \in \mathcal{U}} \max(0, g_v - \alpha_u - d_{u,v}) \leq r_v && \forall v \in \mathcal{V}. && (3) \\ & \alpha_u \geq 0 && \forall u \in \mathcal{U} && \end{aligned}$$

(b) Relaxed Dual

Figure 1: The integer program P (Figure 1a) expresses the Cluster Profit Problem. Lemma 3.1 states that D (Figure 1b) is the non-integer-constrained relaxed dual of P.

Formally, let x_S be a binary variable denoting whether a star S has been picked. Let $\mathcal{T}$ denote the set of all stars. Finding the profit maximizing coalition configuration can then be expressed with the integer program P found in Figure 1.

LEMMA 3.1. *The dual of the non-integer-constrained relaxation of* P *is* D.

One apparent interpretation of the dual variables α_u in D is the profit made by the machine u after paying its distance costs and its share of the resource cost for processing its assigned request.

We now leverage the interpretation of the dual variables to obtain a greedy algorithm to approximately solve the profit maximization problem. Let $g_{\max}$ be the maximum of all the g_v. Informally, the idea behind the algorithm is to start with all dual variables equal to $g_{\max}$ and uniformly decrease their value. Since the α values of all unassigned machines are the same, the maximum profit density star is the star identified by the first equation from Equations (3) that becomes tight. When an equation of type (3) becomes tight, we have given up enough profit to pay for the resource cost of the corresponding request and for the connection costs of the machines to be assigned. Formally, a precise description of the algorithm, which we call ALG is as follows:

Initialization Set each α_u to $g_{\max}$.

Loop Decrease α_u for each machine in $u \in \mathcal{U}$ at a uniform rate until one of the following happens:

1. If α_u becomes zero or $\mathcal{U}$ is empty, then we stop the algorithm.
2. If some inequalities of type (3) become tight, pick one arbitrarily. Say we picked the inequality corresponding to v. Assign all machines in $\mathcal{A} = \{u \mid$

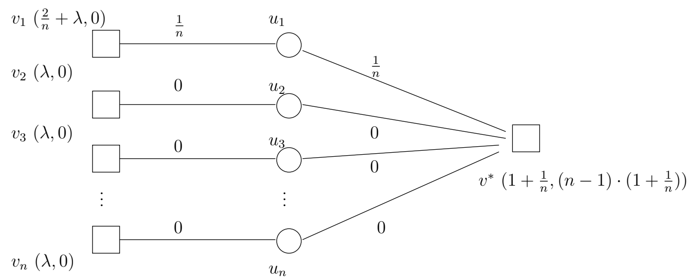

Figure 2: An instance of the Cluster Profit Problem. There are n machines $u_1, \ldots, u_n$. There are $n+1$ requests $v_1, \ldots, v_n$ and v^*. In the figure, each request v is followed by the tuple (g_v, r_v) for the request and the edges are labeled with their distances.

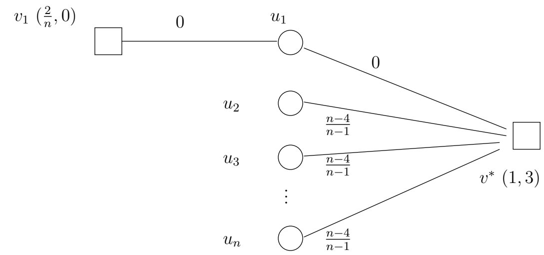

Figure 3: An instance of the Cluster Profit Problem. There are n machines $u_1, \ldots, u_n$. There are 2 requests v_1 and v^*. In the figure, each request v is followed by the tuple (g_v, r_v) for the request and the edges are labeled with their distances. For this example, let n be at least 6.

$g_v - \alpha_u - d_{u,v} \geq 0, u \in \mathcal{U}\}$ to request v. Set $r_v = 0$ and $\mathcal{U} = \mathcal{U} - \mathcal{A}$ and proceed with the uniform decrease once again.

3.1 Two Clarifying Examples

In this section, we run ALG on two specific instances. In both instances, ALG has an approximation ratio of $\Theta(n)$, where n is the number of machines. The first example illustrates the need for the parameter ω. The second example illustrates the need for the parameter θ. Recall the intuitive explanations of these two parameters from Section 2. Let us proceed with the examples.

For the first example, we run ALG on the instance described in Figure 2. In this example we have $\omega = n-1$.

ALG begins by initializing the vector of dual variables α to $1+\frac{1}{n}$, since $g_{\max} = 1+\frac{1}{n}$ in the instance. The algorithm decreases the variables uniformly until they are all equal to $\frac{1}{n}+\lambda$. At this time, the inequality of type (3) for request v_1 becomes tight. It is straight forward to verify that no other inequality of type (3) is tight. The algorithm assigns machine u_1 to request v_1, removes u_1 from $\mathcal{U}$, sets r_{v_1} to zero, and continues decreasing the remaining dual variables uniformly. When the remaining dual variables are all equal to λ, each of the inequalities for v_2 through v_n become tight. Again, it is straight forward to verify that no other inequality is tight. For i from 2 to n, the algorithm assigns machine u_i to request v_i. At this point, all machines are assigned and the algorithm terminates.

The total profit of the solution found by ALG is $\frac{1}{n}+n\cdot\lambda$. However, if we assign all machines to request v^*, then the total profit is 1. Thus, as we let λ go to zero, we find that, for this instance, ALG has an approximation ratio of $\Theta(n)$. It is straight forward to check that ALG would have the same performance even given the rousource augmentation of a constant-factor shorter distances. Thus, the instance in Figure 2, exemplifies the need for the parameter ω.

For the second example, we run ALG on the instance described in Figure 3. In this example we have $\omega = 3$.

ALG begins by initializing the vector of dual variables α to 1, since $g_{\max} = 1$ in the instance. The algorithm decreases the variables uniformly until they are all equal to $\frac{2}{n}$. At this time, the inequality of type (3) for request v_1 becomes tight. The algorithm assigns machine u_1 to request v_1, removes u_1 from $\mathcal{U}$, sets r_{v_1} to zero, and continues decreasing the remaining dual variables uniformly. When the remaining dual variables are all equal to 0, the inequality for v^* becomes tight. No other inequality is tight since n is at least 6. For i from 2 to n, the algorithm assigns machine u_i to request v^*. At this point, all machines are assigned and the algorithm terminates.

The total profit of the solution found by ALG is $\frac{2}{n}$. However, if we assign all machines to request v^*, then the total profit is 1. Thus, we find that for this instance, ALG has an approximation ratio of $\Theta(n)$.

The instance in Figure 3, exemplifies the need for the parameter θ since when we do not give ALG resource augmentation, in other words $\theta = 1$, even given a constant ω, the algorithm has an approximation ratio of $\Theta(n)$. On the other hand, if we give θ-factor shorter distances to ALG, then ALG's solution recovers $\frac{(\theta-1)(n-4)}{\theta} + \frac{2}{n}$ profit in this instance. If θ is a constant then ALG clearly recovers atleast as much profit as the optimal algorithm.

4. ANALYSIS

To analyze ALG, we use a variation of a proof technique called a *factor revealing LP*, which has a lengthy development history but was formalized by Jain et al. [8]. What follows is an overview of our proof structure.

Suppose, as is the case with this paper, we are trying to find an approximation algorithm to an integer constrained maximization problem. Let us call the problem variables π, the problem we are trying to solve P, and its optimal value OPTVAL_P. We write $\text{P}(\pi)$ to denote the objective function value at vector π .

The general approach of a primal-dual algorithm is as follows. First, we relax the integer constraints in the primal program, the problem we are trying to solve, and find the dual of the relaxed problem. Let us call the variables in this dual α, and the dual problem D. Similarly as with P, we write $\text{D}(\alpha)$ to denote the objective function value at vector α. After expressing D, we proceed by finding a real-valued dual variables α' feasible in D and a corresponding integer-valued primal variables π' feasible in P. We hope to find α' and π' in such a way so we can show $\text{D}(\alpha') \leq \gamma \cdot \text{P}(\pi')$. If we succeed, we have γ-approximate algorithm since $\text{P}(\pi') \leq \text{OPTVAL}_\text{P} \leq \text{D}(\alpha') \leq \gamma \cdot \text{P}(\pi')$, where the middle inequality is a result of weak duality. See Vazirani for further details on primal-dual algorithms [22]. The main difficulty when applying the direct primal-dual approach is finding the required mapping from a *feasible* real-valued dual variables α' to a feasible integer-valued primal variables π'.

When applying the factor revealing LP method, we do

$z_k = \min_\mu \max_{g,r,\alpha,d} \mu$
s.t.

$$kg - r - \theta \sum_{i=1}^{k} d_i \le \mu \sum_{i=1}^{k} \alpha_i$$
$$\alpha_i \le \alpha_{i-1} \qquad \forall i \in \{2, ..., k\}$$
$$\alpha_i \le \alpha_j + d_i + d_j \qquad \forall i, j \in \{1, ..., k\}$$
$$\theta d_i \le g \qquad \forall i \in \{1, ..., k\}$$
$$r \le \omega \cdot g$$
$$\sum_{j=i}^{k} \max(0, g - \alpha_i - d_j) \le r \qquad \forall i \in \{1, ..., k\}$$
$$\alpha_i, d_i, r, g \ge 0 \qquad \forall i \in \{1, ..., k\}$$

Figure 4: The definition of z_k. Intuitively, z_k represents the worst approximation factor given that at most k machines contribute to the resource cost of any request (see proof of Lemma 4.4). The definition is required to show Theorem 1, since we take the factor, γ, to be at least $\sup_{i\ge 1}(z_i)$ to show feasibility in $\mathrm{D^L}$.

not insist that α' be feasible D. Instead, we find an π' feasible in P, as in the original primal-dual method, along with a α' which is infeasible in D but has the property that $\mathrm{D}(\alpha') = \mathrm{P}(\pi')$. We proceed by attempting to find a γ, which must be at least 1 by weak duality, such that $\gamma\alpha'$ is feasible in D. We then have $\mathrm{D}(\gamma\alpha') = \gamma\mathrm{D}(\alpha') = \gamma\mathrm{P}(\pi')$, where the first equality comes from the fact that the objective function of D is linear. The vectors π' and $\gamma\alpha'$ have the required properties to finish the original primal-dual argument. Thus, when using the factor revealing LP analysis method, we need not find a complex combinatorial mapping from feasible real-valued dual variables to feasible integer-valued primal variables. Instead we try to find a γ with which we can scale infeasible dual variables so they become feasible. Finding the required γ can often be reduced to solving a series of LPs, which may be easier than finding the required combinatorial mapping. See Jain et al. for further details on the factor revealing LP method [8].

In our work, we must vary the factor revealing LP method slightly, since our results show a competitive approximation ratio. When showing a competitive approximation ratio, we compare our algorithm to an optimal algorithm which has slightly less resources. We use P and D to denote the primal and relaxed dual pair for the instance on which the our algorithm is competing. We will use $\mathrm{P^L}$ and $\mathrm{D^L}$ to denote the primal and relaxed dual pair for the resource hampered instance on which the optimal algorithm is competing. Similar to the factor revealing LP, we find an π' feasible in P and α' infeasible in D such that $\mathrm{P}(\pi') = \mathrm{D}(\alpha')$. However, we then find γ such that $\gamma\alpha'$ is feasible in the optimal algorithm's $\mathrm{D^L}$. We can then show the required $\mathrm{OPTVAL_{P^L}} \le \mathrm{D^L}(\gamma\alpha') = \gamma\mathrm{D^L}(\alpha') = \gamma\mathrm{D}(\alpha') = \gamma\mathrm{P}(\pi')$, where second equality comes from the fact that the objective functions in $\mathrm{D^L}$ and D are the same. The major change from the factor revealing LP method is that we find a scaling γ which makes α' feasible in $\mathrm{D^L}$ instead of D.

By the same reasoning as that for P and D in Section 3 along with the definition of θ from Section 2, we obtain an expression for $\mathrm{D^L}$. The expression is the same as that for D but with occurrences of d replaced by θd.

LEMMA 4.1. *ALG constructs a vector x' feasible in* P *and a vector α' such that* $\mathrm{P}(x') = \mathrm{D}(\alpha')$.

LEMMA 4.2. *Let ALG finish with variables α. For any two machines u, u' and request v, we have $\alpha_u \ge \alpha_{u'} - d_{u,v} - d_{u',v}$.*

LEMMA 4.3. *Let ALG finish with variables α. Consider a request v, any k machines and the corresponding $\alpha_1, \ldots, \alpha_k$, ordered such that $\alpha_{i-1} \ge \alpha_i$. For all $i = 1, \ldots, k$, we have $\sum_{j=i}^{k} \max(0, g_v - \alpha_i - d_{j,v}) \le r_v$.*

LEMMA 4.4. *Let ALG finish with variables α'. Let z_k be as in Figure 4. If $\gamma \ge \sup_{i\ge 1}(z_i)$, then $\gamma\alpha'$ is feasible in* $\mathrm{D^L}$.

PROOF. Fix a request v, and, for a real number μ, consider evaluating $\mu\alpha'$ in the corresponding inequality of $\mathrm{D^L}$, $\sum_{u\in\mathcal{U}} \max(0, g_v - \mu\alpha'_u - \theta d_{u,v}) \le r_v$. Let there be k machines with α'_u such that $g_v - \alpha'_u - \theta d_{u,v} \ge 0$. These are the only machines for which scaling matters. We re-name the machines $\alpha'_1, \ldots, \alpha'_k$ such that $\alpha'_{i-1} \ge \alpha'_i$. Note that , for all k machines, we have that $g_v - \alpha'_u - \theta d_{u,v} \ge 0$ which implies that $g_v \ge \theta d_{u,v}$, since α'_u is always at least 0.

We rewrite the inequality in the previous paragraph using the naming scheme we have described to get $\sum_{i=1}^{k}(g_v - \mu\alpha'_i - \theta d_{i,v}) \le r_v$. We rearrange the inequality to get $kg_v - r_v - \theta\sum_{i=1}^{k} d_{i,v} \le \mu\sum_{i=1}^{k}\alpha'_i$.

Let μ' be the minimum μ which satisfies the above equation. In the next paragraph, we argue that the values μ', g_v, r_v, $\alpha'_1, \ldots, \alpha'_k$, $d_{1,v}, \ldots, d_{k,v}$ are feasible in z_k. Given this feasibility, since $\gamma \ge \sup_{i\ge 1}(z_i)$, we have $\gamma \ge z_k \ge \mu'$. Thus, $kg_v - r_v - \theta\sum_{i=1}^{k} d_{i,v} \le \mu'\sum_{i=1}^{k}\alpha'_i \le \gamma\sum_{i=1}^{k}\alpha'_i$.

Thus, $\gamma\alpha'$ is feasible in the inequality for v in $\mathrm{D^L}$. The same argument holds for all requests v, thus $\gamma\alpha'$ is feasible in $\mathrm{D^L}$.

All that remains to be shown is that the vector of values from the preceding paragraph is feasible in z_k. The values are feasible in the first inequality of the definition of z_k by the definition of μ'. They are feasible in the second constraint by our ordering on $\alpha'_1, \ldots, \alpha'_k$. Feasibility in the third constraint comes from Lemma 4.2. Feasibility in the fourth inequality comes from the fact that for all k machines we have $g - \alpha_i - \theta d_i \ge 0$. Feasibility in the fifth inequality comes from our constraints on what requests can come into the system, in other words, the definition of ω. Feasibility in the final constraint comes from Lemma 4.3. The non-negativity constraints are also satisfied by the definition of CPP and ALG. $\square$

LEMMA 4.5. *Let z_k be as in Figure 4. The value of z_k is at most k.*

LEMMA 4.6 (MAIN TECHNICAL LEMMA). *Let z_k be as in Figure 4. There exist constants γ^* and k^*, only dependant on the constants ω and θ, such that* $\sup_{k\ge k^*}(z_k) \le \gamma^*$.

LEMMA 4.7. *Let ALG finish with variables α. Then, there exists a constant γ, only dependent on the constants ω and θ, such that $\gamma\alpha$ is feasible in* $\mathrm{D^L}$.

PROOF. By Lemma 4.4, if $\gamma \ge \sup_{i\ge 1}(z_i)$ then $\gamma\alpha$ is feasible in $\mathrm{D^L}$. Let γ^* and k^* be constants as in Lemma

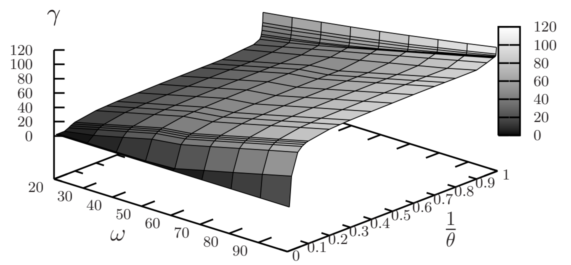

Figure 5: The exact competitiveness factor, γ, as a function of the parameters ω and θ. In the experiments, ω was varied between 20 and 100, while θ was varied between 0 and 1. As θ goes to zero, the approximation factor γ approaches one. As θ goes to one, γ approaches infinity. For the mid-ranges of θ, the approximation factor γ varies almost linearly with ω. In other words, as ω goes to one, so does γ; as ω goes to infinity, so does γ.

4.6. Then, we have $\sup_{i\geq 1}(z_i) \leq \max(z_1, \ldots, z_{k^*}, \gamma^*)$. By Lemma 4.5, we have $\sup_{i\geq 1}(z_i) \leq \max(k^*, \gamma^*)$. Thus, setting γ to the constant $\max(k^*, \gamma^*)$ gives the desired result. □

THEOREM 1. *For a constant γ, ALG operating on an instance of CPP is γ-competitive against an optimal algorithm operating on the same instance, but with distances increased by a constant factor θ.*

PROOF. By Lemma 4.1, ALG constructs a vector x feasible in P and a vector α such that $\mathrm{P}(x) = \mathrm{D}(\alpha)$. By Lemma 4.7, there exists a constant γ such that $\gamma\alpha$ is feasible in $\mathrm{D^L}$. Let $\mathrm{OPTVAL_{PL}}$ be the value of the optimal algorithm. By weak duality we have $\mathrm{OPTVAL_{PL}} \leq \mathrm{D^L}(\gamma\alpha)$. By the linearity of the objective function of $\mathrm{D^L}$, we have $\mathrm{D^L}(\gamma\alpha) = \gamma\mathrm{D^L}(\alpha)$. Since the objective function of D and $\mathrm{D^L}$ are the same, we have $\gamma\mathrm{D^L}(\alpha) = \gamma\mathrm{D}(\alpha) = \gamma\mathrm{P}(x)$. Thus, we have shown the desired result. □

Using Lemma 4.6, given values for θ and ω, it is possible to numerically compute the minimum value of γ for which ALG is γ-competitive. The results of such a numerical computation are displayed in Figure 5.

5. RESOURCE AUGMENTED STABILITY

In this section, we prove game theoretic stability results for ALG.

5.1 Standard Game Theoretic Definitions

In coalitional game theory, a game is usually defined by a *characteristic function* $V : 2^{\mathcal{P}} \to \Re_+$, where $\mathcal{P}$ is the set of players and $\Re_+$ are the non-negative real numbers [20]. In a profit maximizing game, $V(\mathcal{A})$ represents the maximum amount of profit the players in $\mathcal{A}$ can guarantee themselves, without the cooperation of the remaining players.

A payoff vector π is in the *core* of a coalitional game if it satisfies two types of inequalities. First, the vector must satisfy the *stability inequalities* $\sum_{a\in\mathcal{A}} \pi_a \geq V(\mathcal{A})$ for all $\mathcal{A} \subseteq \mathcal{P}$. Second, the vector must satisfy the *conservation inequality* $\sum_{a\in\mathcal{P}} \pi_a \leq V(\mathcal{P})$.

The stability inequality for $\mathcal{A}$ demands that the players $\mathcal{A}$ receive at least as much payoff in vector π as the subset could receive playing optimally on their own. Thus, for a payoff vector in the core, no subset of the players has incentive to reject the payoff vector and play on its own.

The conservation inequality demands that the vector π does not give away more payoff than the game allows the entire set of players to receive. See [20] for more details on the core of a coalitional game.

5.2 The Cluster Coalition Game

The Cluster Profit Problem gives rise to a natural coalitional game, which we call the Cluster Coalition Game. The players of the game are the set of machines $\mathcal{U}$, and the characteristic function $V(\mathcal{A})$ is the maximum amount of profit the machines in $\mathcal{A}$ can achieve working alone.

We can express $V(\mathcal{A})$ using the terminology of Section 3. For a subset of machines $\mathcal{A}$, let $\mathcal{T}_{\mathcal{A}}$ denote the stars which include only machines from $\mathcal{A}$. We can then write $V(\mathcal{A})$ with the same integer program as P, but with $\mathcal{T}$ replaced with $\mathcal{T}_{\mathcal{A}}$ (see Figure 1).

Goemans et al. have shown that the core of a related Facility Location Game is often empty [6]. However, even if the core of the Cluster Coalition Game were non-empty, it would be NP-hard to find a core vector. Notice that the core conservation inequality along with the stability inequality for $\mathcal{A} = \mathcal{P}$ imply that $\sum_{a\in\mathcal{P}} \pi_a = V(\mathcal{P})$. On the other hand, Lemma B.1 shows that determining $V(\mathcal{P})$ is NP-hard. Thus, finding a vector in the core of the Cluster Coalition Game is also NP-hard.

5.3 RA Stability Definition

As described in Section 5.2, there is little hope of finding a vector in the core of the Cluster Coalition Game. However, we can still show stability properties for the coalitions found by ALG. In this section, we give a natural definition of RA stability.

Fix a profit maximization game parameterized by a vector x and defined by the characteristic function V^x. A payoff vector π is γ-*RA stable* under a transformation L if it satisfies the following inequalities. First, the vector must satisfy the *modified stability inequalities* $\sum_{a\in\mathcal{A}} \pi_a \geq \frac{1}{\gamma} V^{\mathrm{L}(x)}(\mathcal{A})$ for all $\mathcal{A} \subseteq \mathcal{P}$. Second, the vector must satisfy the conservation inequality $\sum_{a\in\mathcal{P}} \pi_a \leq V^x(\mathcal{P})$.

Notice that this definition parallels the definition of a RA approximation algorithm from Section 2. The transformation L decreases some resource in the system. For notational brevity, we assume the parameter x to be implicit and from here on write V^{L} and V instead of $V^{\mathrm{L}(x)}$ and V^x, respectively.

We remind the reader of the example in Section 1.1. If the machines in the example deviate then they cannot use the company's proprietary network and incur larger communication costs. If the coalitions suggested by the company, and resulting payoff vector, are 1-RA stable, then the company is assured that no subset of the machines has incentive to deviate. If the suggested coalitions are γ-RA stable, where γ is a small constant, the company is assured that no subset of machines has a large incentive to deviate.

5.4 RA Stability for Cluster Coalition

In this section we show that the coalitions computed by ALG are γ-RA stable, for some constant γ with respect to lengthening the graph distances by a constant factor.

Let the Cluster Coalition Game defined in Section 5.2 be parameterized with the distance metric d. For our proof, the transformation L lengthens the distances by a constant factor θ. Thus, we define $V^{\mathrm{L}}(\mathcal{A})$ with the same linear program as $V(\mathcal{A})$, where the profit of each star, p_S^{L}, is computed in a graph with a distance metric θd.

THEOREM 2. *Let ALG compute a vector of dual variables α. For a constant γ, the payoff vector α is γ-RA stable in the Cluster Coalition Game, with respect to lengthening the graph distances by a constant θ.*

PROOF. The crucial part of the proof is Lemma 5.1. □

LEMMA 5.1. *If a vector α is feasible in the non-integer-constrained relaxed dual for $V^{\mathrm{L}}(\mathcal{P})$, it is also feasible in the non-integer-constrained relaxed dual for $V^{\mathrm{L}}(\mathcal{A})$ for any $\mathcal{A} \subseteq \mathcal{P}$.*

6. REFERENCES

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, U.K., 1998.

[2] M. Charikar, S. Khuller, D. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the 12th annual ACM-SIAM Symposium on Discrete algorithms*, pages 642–651, Washington, DC, January 2001.

[3] B.G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. Papadimitriou, and J. Kubiatowicz. Selfish caching in distributed systems: a game-theoretic analysis. In *Proceedings of the 23rd annual ACM Symposium on Principles of Distributed Computing*, pages 21–30, St. John's, Newfoundland, July 2004.

[4] Z. Drezner and H. W. Hamacher. *Facility Location: applications and theory*. Springer, New York, 2002.

[5] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *Proceedings of the 22nd annual symposium on Principles of Distributed Computing*, pages 347–351, Boston, MA, July 2003.

[6] M. X. Goemans and M. Skutella. Cooperative facility location games. *Journal of Algorithms*, 50:194–214, 2004.

[7] D. S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22:148–162, 1982.

[8] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing lp. *J. ACM*, 50(6):795–824, 2003.

[9] K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problem. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 2, New York City, October 1999.

[10] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[11] L. Kaufman, M.V. Eede, and P. Hansen. A plant and warehouse location problem. *Operational Research Quarterly*, 28:547–554, 1977.

[12] I. Keider, R. Melamed, and A. Orda. Equicast: scalable multicast with selfish users. In *Proceedings of the 25th annual symposium on Principles of Distributed Computing*, pages 63–71, Denver, Colorado, USA, July 2006.

[13] R. R. Mettu and C. G. Plaxton. The online median problem. *SIAM Journal on Computing*, 32:816–832, 2003.

[14] Thomas Moscibroda and Roger Wattenhofer. Facility location: distributed approximation. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 108–117, 2005.

[15] M. Pál and E. Tardos. Group strategy proof mechanisms via primal-dual algorithms. In *Proceedings of the 44th annual IEEE Symposium on Foundations of Computer Science*, pages 584–593, October 2003.

[16] Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 140–149, 1997.

[17] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111:209–238, March 1999.

[18] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. *International Joint Conference on Artificial Intelligence*, 14:655–661, August 1995.

[19] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th annual ACM symposium on Theory of Computing*, pages 265–274, El. Paso, Texas, USA, July 1997.

[20] M. Shubik. *Game Theory In The Social Sciences*. MIT Press, Cambridge, Massachussetts, 1984.

[21] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[22] V. V. Vazirani. *Approximation Algorithms*. Springer, New York, 2001.

APPENDIX

A. ALGORITHM DESCRIPTION

PROOF OF LEMMA 3.1. A relaxation of P can be obtained by substituting Equation (2) with $x_S \geq 0$. We do not require the upper bound $x_S \leq 1$, since it is already imposed by Equation (1).

We take the dual of the relaxed program.

$$\begin{aligned} \min \quad & \sum_{u \in \mathcal{U}} \alpha_u \\ \text{s.t.} \quad & \sum_{u \in S} \alpha_u \geq p_S \qquad \text{for all } S \in \mathcal{T} \\ & \alpha_u \geq 0 \qquad\qquad \text{for all } u \in \mathcal{U} \end{aligned}$$

We re-write Equation (4) several times to show the desired result. First, we begin by substituting the definition of p_S.

$$\sum_{u \in S} \alpha_u \geq |S| \cdot g_{v_S} - \sum_{u \in S} d_{u,v_S} - r_{v_S} \qquad \text{for all } S \in \mathcal{T}.$$

Re-arranging, we get $\sum_{u \in S}(g_{v_S} - \alpha_u - d_{u,v_S}) \leq r_{v_S}$ for all $S \in \mathcal{T}$. The result then follows from Lemma A.1. □

LEMMA A.1. *The inequalities*

$$\sum_{u \in S}(g_{v_S} - \alpha_u - d_{u,v_S}) \leq r_{v_S} \qquad \text{for all } S \in \mathcal{T}. \tag{4}$$

and the inequalities specified by Equations (3) are equivalent.

PROOF. To see that any vector α satisfying Equations (4) also satisfies Equations (3), for each request v consider the star $S = (v, \mathcal{A})$ with $\mathcal{A} = \{u \mid g_v - \alpha_u - d_{u,v} \geq 0\}$. For the reverse direction, notice that $\sum_{u \in S}(g_{v_S} - \alpha_u - d_{u,v_S})$ is at most $\sum_{u \in \mathcal{U}} \max(0, g_{v_S} - \alpha_u - d_{u,v_S})$. □

B. ANALYSIS

PROOF SKETCH FOR LEMMA 4.1. The solution of ALG is the assignment of machines to stars. Each variable α_u can be interpreted as the profit of machine u, after paying its distance to its assigned request and its share of the computational resource expense. We only assign machines to requests when the corresponding inequality of type (3) is tight. Thus, at the time of assignment for a request, the algorithm has deducted enough to pay for the resource expense of the request. The lemma can be fully formally proved by induction on the number of machine to request assignments by the algorithm. □

PROOF OF LEMMA 4.2. If $\alpha_u \geq \alpha_{u'}$, we are done.

Assume $\alpha_u < \alpha_{u'}$. In other words, machine u' is connected to a request, say v', before α_u stops decreasing.

If $g_{v'} - \alpha_u - d_{u,v'} > 0$, then machine u should connect to request v' before we lower $r_{v'}$ to zero. In other words, u must be in the first set of machines which we connect to v', and we must have $\alpha_u \geq \alpha_{u'}$, which is a contradiction with the assumption in the previous paragraph.

Thus, we have

$$g_{v'} - \alpha_u - d_{u,v'} \leq 0$$
$$g_{v'} - d_{u,v} - d_{u',v} - d_{u',v'} \leq \alpha_u,$$

where the last statement comes from the triangle inequality. Since u' connects to v', we have $g_{v'} - \alpha_{u'} - d_{u',v'} \geq 0$ which we can then use to continue re-writing

$$g_{v'} - d_{u',v'} - d_{u,v} - d_{u',v} \leq \alpha_u$$
$$\alpha_{u'} - d_{u,v} - d_{u',v} \leq \alpha_u$$

□

PROOF OF LEMMA 4.3. Assume $\sum_{j=i}^{k} \max(0, g_v - \alpha_i - d_{j,v}) > r_v$.

If for all $l \in \{i, \ldots, k\}$, we have $\alpha_l \leq \alpha_i$, then all machines in the set $\{i, \ldots, k\}$ are unassigned when ALG has decreased α to α_i. But, this is a contradiction since ALG would assign the entire set of machines $\{i, \ldots, k\}$ to v when the α are at $\lambda > \alpha_i$ such that $\sum_{j=i}^{k} \max(0, g_v - \lambda - d_{j,v}) = r_v$.

Thus, there exists $l \in \{i, \ldots, k\}$, such that $\alpha_l > \alpha_i$. But, this is a contradiction with the ordering on $\alpha_1, \ldots, \alpha_k$. □

PROOF OF LEMMA 4.5. We would like to show that regardless of the setting of g, r, α, d, assigning k to μ is always feasible in z_k.

From the sixth constraint in z_k, we have

$$\begin{aligned} r &\geq \sum_{j=1}^{k} \max(0, g - \alpha_1 - d_j) \\ &\geq \sum_{j=1}^{k} (g - \alpha_1 - d_j) \\ &\geq \sum_{j=1}^{k} (g - \sum_{i=1}^{k} \alpha_i - d_j) \\ &= kg - k\sum_{i=1}^{k} \alpha_i - \sum_{j=1}^{k} d_j) \\ &\geq kg - k\sum_{i=1}^{k} \alpha_i - \theta \sum_{j=1}^{k} d_j, \end{aligned}$$

where the last inequality comes from the fact that $\theta \geq 1$.

Rearranging, we have

$$kg - r - \theta \sum_{j=1}^{k} d_j \leq k \sum_{i=1}^{k} \alpha_i.$$

Thus, setting μ to k is always feasible in z_k. □

PROOF OF LEMMA 4.6. Suppose we are trying to solve for z_k. We would like to find a μ, such that for all settings of the remaining variables,

$$\frac{k}{\mu} g - \frac{1}{\mu} r - \frac{\theta}{\mu} \sum_{i=1}^{k} d_i \leq \sum_{i=1}^{k} \alpha_i, \tag{5}$$

We find a setting for μ by using the inequalities imposed on g, r, α, d. One of these inequalities is

$$\sum_{j=i}^{k} \max(0, g - \alpha_i - d_j) \leq r \qquad \text{for all } i \text{ in } \{1, ..., k\},$$

which we can relax to

$$\sum_{j=i}^{k} (g - \alpha_i - d_j) \leq r \qquad \text{for all } i \text{ in } \{1, ..., k\}.$$

Multiplying the inequality for i by a scaling factor ϕ_i and summing all the resulting inequalities, we have

$$\sum_{i=1}^{k} \phi_i \sum_{j=i}^{k} (g - \alpha_i - d_j) \leq r \sum_{i=1}^{k} \phi_i.$$

Expanding the double summation on the left hand side and re-arranging, we have

$$g \sum_{i=1}^{k} \phi_i (k - i + 1) - r \sum_{i=1}^{k} \phi_i - \sum_{i=1}^{k} d_i \sum_{j=1}^{i} \phi_j \leq \sum_{i=1}^{k} \alpha_i \phi_i (k - i + 1). \tag{6}$$

Recalling that our goal is to show Inequality (5), we can now see that it is reasonable to set ϕ_i to $\frac{1}{(k-i+1)}$ in Inequality

(6). With these values, Inequality (6) reduces to

$$kg - r\sum_{i=1}^{k}\frac{1}{k-i+1} - \sum_{i=1}^{k} d_i \sum_{j=1}^{i}\frac{1}{k-i+1} \leq \sum_{i=1}^{k}\alpha_i. \quad (7)$$

We proceed with some intuition. Comparing Inequality (7) to the goal Inequality (5), we notice that there is some slack in the coefficient of g, but the coefficients of r and d_i are too large. Thus, we cannot yet claim we have shown the desired result. Instead, we will use the inequalities $r \leq \omega g$ and $\theta d_i \leq g$ for all i in $\{1, ..., k\}$, which must hold by the definition of z_k, to siphon some of the excess coefficient of g to reduce the coefficients of r and d_i.

More formally, we introduce non-negative parameters s_r, $s_{d_1}, \ldots s_{d_k}$, each representing the amount of g coefficient which we will siphon to the specified destination. Using the two inequalities in the above paragraph and Inequality (7), we have

$$\begin{aligned}
&(k - s_r - \sum_{i=1}^{k} s_{d_i})g - r(-\frac{s_r}{\omega} + \sum_{i=1}^{k}\frac{1}{k-i+1}) \\
&\quad - \sum_{i=1}^{k} d_i(-\theta \cdot s_{d_i} + \sum_{j=1}^{i}\frac{1}{k-i+1}) \\
&\quad \leq kg - r\sum_{i=1}^{k}\frac{1}{k-i+1} - \sum_{i=1}^{k} d_i \sum_{j=1}^{i}\frac{1}{k-i+1} \leq \sum_{i=1}^{k}\alpha_i.
\end{aligned}$$

Thus, as long as we can find a setting of the variables s satisfying

$$\begin{aligned}
k - s_r - \sum_{i=1}^{k} s_{d_i} &\geq \frac{k}{\mu} \\
-\frac{s_r}{\omega} + \sum_{i=1}^{k}\frac{1}{k-i+1} &\leq \frac{1}{\mu} \\
-\theta \cdot s_{d_i} + \sum_{j=1}^{i}\frac{1}{k-i+1} &\leq \frac{\theta}{\mu} \qquad \text{for all } i \text{ in } \{1, \ldots, k\}
\end{aligned}$$

we would show that $z_k \leq \mu$. We proceed by simplifying the above expression and finding values of μ, k for which it holds, given constants ω and θ.

We simplify the inequalities in the previous paragraph, by setting $s_d = \sum_{i=1}^{k} s_{d_i}$ and summing the last inequality over all i. We must then more simply show there exist s_r and s_d such that

$$\begin{aligned}
k - s_r - s_d &\geq \frac{k}{\mu} \\
-\frac{s_r}{\omega} + \sum_{i=1}^{k}\frac{1}{k-i+1} &\leq \frac{1}{\mu} \\
-\theta \cdot s_d + \sum_{i=1}^{k}\sum_{j=1}^{i}\frac{1}{k-i+1} &\leq \frac{k\theta}{\mu}
\end{aligned}$$

Noting that $\sum_{i=1}^{k}\sum_{j=1}^{i}\frac{1}{k-i+1} = k$ and $\sum_{i=1}^{k}\frac{1}{k-i+1} \leq 1 + \log k$, we can further simplify and strengthen the inequalities we must satisfy to

$$\begin{aligned}
k - s_r - s_d &\geq \frac{k}{\mu} \\
-\frac{s_r}{\omega} + (1 + \log k) &\leq \frac{1}{\mu} \\
-\theta \cdot s_d + k &\leq \frac{k\theta}{\mu}
\end{aligned}$$

Setting $s_r = s_d = \frac{k}{2}(1 - \frac{1}{\mu})$ satisfies the first inequality. If we set μ to any value γ^* such that

$$(1 - \frac{1}{\gamma^*})\frac{\theta}{2} \geq 1,$$

we satisfy the third inequality. Finally, the middle inequality is then satisfied for any $k \geq k^*$ where k^* is the least integer satisfying

$$\frac{1 - \frac{1}{\gamma^*}}{2\omega}k^* \geq 1 + \log k^*$$

Thus, we have shown $z_k \leq \gamma^*$ for any $k \geq k^*$. □

Lemma B.1. *The Cluster Profit Problem is NP-Hard.*

Proof. We prove this by reducing the Facility Location Problem (FLP) to CPP. It is known that FLP is NP-Hard. Let an instance of FLP be given by the set of facilities ,$\mathcal{F}$; the set of clients ,$\mathcal{J}$; the cost for opening each facility $p \in \mathcal{F}$, f_p; and the cost for connecting a client $j \in \mathcal{J}$ to facility p, $c_{j,p}$. We create an instance of CPP by letting the set of requests and machines be $(\mathcal{V}, \mathcal{U}) = (\mathcal{F}, \mathcal{J})$. We define the resource cost of a request p as $r_p = f_p$. The distance cost of a machine j to a request p is given by $d_{j,p} = c_{j,p}$. Let g be a constant greater than the sum of the maximum facility opening cost and the maximum connection cost. For each request p we set $g_p = g$. The value of g ensures that for each machine it is more profitable to work on any request than not work at all.

In this instance of CPP, regardless of what requests are serviced the total revenue generated is $g \cdot |\mathcal{U}|$. So the optimal solution to CPP instance is the one that minimizes the overall resource and connection costs. Therefore the solution of CPP instance is also a solution to the FLP instance. □

C. RA STABILITY

Lemma C.1. *Let ALG compute a vector of dual variables α. Then, $\sum_{a \in \mathcal{P}} \alpha_a \leq V(\mathcal{P})$*

Proof of Lemma C.1. Recall from Section 5.2 that the set of players, $\mathcal{P}$, is the set of machines $\mathcal{U}$.

By Lemma 4.1, ALG constructs a vector x feasible in P and a vector α such that $\mathrm{P}(x) = \mathrm{D}(\alpha)$. Since the program for P from Section 3 and $V(\mathcal{P})$ from Section 5.2 is the same, we have

$$\sum_{a \in \mathcal{P}} \alpha_a = \mathrm{D}(\alpha) = \mathrm{P}(x) \leq V(\mathcal{P}).$$

The first equality comes from the objective function of D. The second equality comes from Lemma 4.1. The last inequality comes from the fact that x is feasible in P which has optimal objective function value $V(\mathcal{P})$. □

Lemma C.2. *Let ALG compute a vector of dual variables α. There is a constant γ such that for any $\mathcal{A} \subseteq \mathcal{P}$, we have $\sum_{a \in \mathcal{A}} \alpha_a \geq \frac{1}{\gamma} V^{\mathrm{L}}(\mathcal{A})$.*

PROOF OF LEMMA C.2. Recall that in the Cluster Coalition Game, the set of players $\mathcal{P}$ is the set of machines $\mathcal{U}$.

By Lemma 4.7, we have that there exists a constant γ such that $\gamma\alpha$ is feasible in $\mathrm{D^L}$. But, $\mathrm{D^L}$ is also the expression for the relaxed dual of $V^{\mathrm{L}}(\mathcal{P})$. Thus, we have

$$\gamma \sum_{a\in\mathcal{P}} \alpha_a = \gamma \mathrm{D^L}(\alpha) = \mathrm{D^L}(\gamma\alpha) \geq V^{\mathrm{L}}(\mathcal{P}).$$

The first equality comes from the objective function of $\mathrm{D^L}$. The second equality comes from the linearity of the objective function of $\mathrm{D^L}$. The third inequality comes from the fact that $\gamma\alpha$ is feasible in $\mathrm{D^L}$ and $\mathrm{D^L}$ is the non-integer constrained relaxed dual of $V^{\mathrm{L}}(\mathcal{P})$.

Lemma 5.1 allows us to apply the same reasoning for any $\mathcal{A} \subseteq \mathcal{P}$, $\square$

PROOF OF LEMMA 5.1. Recall that in the Cluster Coalition Game, the set of players $\mathcal{P}$ is the set of machines $\mathcal{U}$. Let α be feasible in $\mathrm{D^L}$, which is the relaxed dual of $V^{\mathrm{L}}(\mathcal{P})$

The expression for the relaxed dual of $V^{\mathrm{L}}(\mathcal{A})$ is

$$\begin{aligned} \min \quad & \sum_{u\in\mathcal{A}} \alpha_u \\ \text{s.t.} \quad & \sum_{u\in\mathcal{A}} \max(0, g_v - \alpha_u - \theta d_{u,v}) \leq r_v \qquad \forall v \in \mathcal{V}. \\ & \alpha_u \geq 0 \qquad \forall u \in \mathcal{A}, \end{aligned}$$

For $\mathcal{A} \subseteq \mathcal{U}$, α is feasible in the above program since $\mathcal{A} \subseteq \mathcal{U}$ implies

$$\begin{aligned} & \sum_{u\in\mathcal{A}} \max(0, g_v - \alpha_u - \theta d_{u,v}) \\ & \quad \leq \sum_{u\in\mathcal{U}} \max(0, g_v - \alpha_u - \theta d_{u,v}) \leq r_v. \end{aligned}$$

$\square$

C.1 Distributed Algorithm

The main purpose of this section is to describe a distributed implementation of ALG.

In ALG we attached a profit variable to each of the machines and let it reduce uniformly. However for the distributed algorithm it is more intuitive to think of balls being grown around each of the requests. We define a ball $B_v(R)$ to be the set of machines that are within a distance R from the request v. For each machine u, we call a request v *local* if $u \in B_v(g_v)$; correspondingly u is a *local* machine for v. Throughout the distributed algorithm, let $A_v(R)$ denote the set of unassigned, local machines that are within distance R from v.

We proceed with a short sketch of the distributed algorithm. Initially, we label all requests as undecided. The distributed algorithm proceeds in phases. Each phase is as follows. Every undecided request makes a profit offer to some of its local machines. Each machine accepts the highest profit offer received. If for a request v, all recipients of v's offer accept, we switch the label of v from undecided to open. We then proceed to the next phase.

Formally, each machine u maintains three lists: *Opened* local requests, $\mathcal{O}_u$; *Closed* local requests, $\mathcal{C}_u$; and *Undecided* local requests, $\mathcal{U}_u$. In addition, each machine keeps a variable P_v for each local request v that denotes the profit offer it has received from v. The value of P_v maybe updated as the algorithm proceeds in phases. We define P_u^{max} be the maximum profit offer that machine u has from any local *opened* request, i.e. $P_u^{max} = \max_{v\in\mathcal{O}_u}(P_v)$.

Each requests v maintains the set $A_v(g_v)$. For any unaccepted request v, we define a *safe-machine list*, $\mathcal{M}_v$, and a *safe fill-radius*,t_v, such that $\mathcal{M}_v = \{u \mid u \in A_v(t_v), g_v - t_v \geq P_u^{max}\}$ and $\sum_{u\in\mathcal{M}_v}(t_v - d_{u,v}) = r_v$ and $t_v \leq g_v$.

Sometimes, no such pair of $\mathcal{M}_v$ and t_v may exist. This is by design, since in such cases the request v is moved to the closed list of all local machines.

In the distributed algorithm, the request v makes an offer of t_v to the machines in $\mathcal{M}_v$. Each machine $u \in \mathcal{M}_v$, can then calculate its profit when working on v as $P_v = g_v - t_v$. The definition of safe-fill radius ensures that every machine in $\mathcal{M}_v$ receives more profit from working on v than from working on any local, *opened* request.

We now give the distributed algorithm for the Cluster Profit Problem. Each phase consists of several rounds. We annotate a round i occurring at the request as REQ_i and the one occurring at the machine as MAC_i. The distributed algorithm is as follows:

Initialization Every request v, sends g_v to its local machines. Each machine sends its distance value $d_{u,v}$ to all its local requests and sets $P_v = g_v$

Phase The algorithm proceeds in phases until each machine is either assigned or cannot be assigned because all its local requests are in the *Closed* list. Each phase has the following five rounds:

REQ_1: Each undecided request v calculates the *safe-fill radius*, t_v. If no safe-fill radius exists, then v sends $\langle Close\rangle_v$ to machines in $B_v(g_v)$ and stops further communication. Otherwise, v sends t_v to machines in $B_v(g_v)$.

MAC_1: If a machine u receives a $\langle Close\rangle_v$ then it puts v in the *Closed* list. Otherwise, it updates the corresponding profit variable to $P_v = g_v - t_v$. Let $v' = \max_v\{P_v | v \in \mathcal{U}_u \cup \mathcal{O}_u\}$. Machine u sends an $\langle Accept\rangle_u$ message to v' and a $\langle Reject\rangle_u$ message to all other local requests.

REQ_2: If a request v receives all $\langle Accept\rangle_u$ messages from $u \in \mathcal{M}_v$, then it has enough machines to pay for its costs so it sends $\langle Open\rangle_v$ to all local machines, and stops further communication.

MAC_2: If a machine u receives $\langle Open\rangle_v$ then it puts v to its *Opened* list and sets $P_v = g_v - max(d_{u,v}, t_v)$. If $u \in \mathcal{M}_v$ then it gets assigned to v and sends $\langle Inactive\rangle_u$ to all other local requests. If u receives no messages in this round then it gets assigned to the request v in its *Opened* list that has the maximum P_v.

REQ_3: If a request v receives $\langle Inactive\rangle_u$ then it removes u from $A_v(g_v)$.

It is not difficult to show the two following results to describe the communication complexity of the destributed algorithm.

LEMMA C.3. *In each phase at least one undecided request is either opened or closed.*

THEOREM 3. *The distributed algorithm finishes in $O(|\mathcal{V}|)$ phases. Each machine exchanges $O(|\mathcal{V}|^2)$ local messages and the total number of local messages exchanged during the execution of the algorithm is $O(|\mathcal{U}||\mathcal{V}|^2)$.*