Decentralized Algorithms for Distributed Integer Programming Problems with a Coupling Cardinality Constraint

Ezgi Karabulut, Shabbir Ahmed, George Nemhauser

Abstract

We consider a multi-player optimization where each player has her own optimization problem and the individual problems are connected by a cardinality constraint on their shared resources. We give distributed algorithms that allow each player to solve their own optimization problem and still achieve a global optimization solution for problems that possess a concavity property. For problems without the concavity property, we use concave approximating functions to bound the optimality error and provide empirical results on the deviations from optimality.

1 Introduction

Solving applications involving large and complex data sets, frequently requires decentralized storage of data and distributed algorithms. This need has given rise to the field of distributed optimization, where an optimization problem is solved with multiple processors (interchangeably referred to as *agents* or *players* throughout the paper) communicating with each other without the need of a central coordinator. Research in this area focuses on how to distribute the data to increase algorithmic efficiency, how to select a smaller subset of the data to solve the problem with minimal loss of accuracy, or how to parallelize the algorithm to make it more efficient. Distributed optimization may also arise in the setting of multiple players who are cooperating to solve a common problem but do not want to fully share their private data hence making it impossible to solve the problem with a centralized processor. The challenge is to solve these problems with high accuracy and limited communication when the data is distributed among multiple processors.

Much of the work on distributed optimization has focused on convex optimization. To the best of our knowledge, there has been very limited progress in the discrete setting. In this paper we consider multi-player discrete optimization problems where the players share a single resource. We design decentralized algorithms that do not require a central processor to allocate the resource optimally across the players to solve the overall problem.

The problem that we investigate (P1) has the structure of independent problems coupled together with a knapsack constraint. Independent players sharing a common resourse is a prevalent phenomenon. Many budgeting problems involving multiple players are of this form. Some applications of this problem can be found in areas such as network optimization and portfolio optimization. Without loss of generality, we assume all data is integer.

$$z = \max\left\{\sum_{i=1}^{m} f_i(x_i) : \sum_{i=1}^{m} a_i^T x_i \leq K \ , \ x_i \in X_i \ , \ x_i \in \{0,1\}^{N_i} \quad i = 1, \ldots, m\right\} \tag{P1}$$

Here, $i = \{1, \ldots, m\}$ is the set of players, x_i is the binary decision vector, X_i is the set of local constraints, and $f_i : \mathbf{B}^{N_i} \to \mathbf{R}$ is the objective function for players $i = \{1, \ldots, m\}$ respectively. Given an allocation of the resource to the players, i.e. a collection $\{K_i\}_{i=1}^m$ such that $\sum_{i=1}^m K_i = K$, (P1) can be partitioned into subproblems:

$$z_i(K_i) = \max\left\{ f_i(x_i) : a_i^T x_i \le K_i \ , \ x_i \in X_i \ , \ x_i \in \{0,1\}^{N_i} \right\}. \qquad (\mathrm{P1}_i)$$

Clearly, any arbitrary collection $\{K_i\}_{i=1}^m$ with $\sum_{i=1}^m K_i = K$, satisfies $\sum_{i=1}^m z_i(K_i) \le z$. Furthermore, there exists an optimal allocation $\{K_i^*\}_{i=1}^m$ that satisfies $\sum_{i=1}^m K_i^* = K$ for which $\sum_{i=1}^m z_i(K_i^*) = z$. This means that when we have the optimal allocation $\{K_i^*\}_{i=1}^m$ of the resource, each player can solve their subproblem (P1$_i$) independently, and still find the optimal solution of (P1). In this paper, we propose decentralized algorithms that exploit this property, and seek an optimal allocation of K to solve (P1) in a distributed manner with limited information sharing. We derive specialized algorithms designed for the case where the coupling constraint is a cardinality constraint.

Distributed algorithms have been widely used in convex optimization. Studies by Nedic and Ozdaglar [17], Lobel et al. [15] and Raffard et al. [20] cover the distributed version of the subgradient method. Duchi et al. [7] propose distributed algorithms based on dual subgradient averaging. Wei and Ozdaglar [24] and Boyd et al. [5] provide analysis for distributed ADMM algorithms. Jadbabaie et al. [9] study an alternative approach to Newton-type distributed algorithms. Recent research by Jakovetic et al. [10] includes a distributed gradient algorithm based on the Nesterov gradient.

Much less research has been done on distributed optimization of discrete problems. Mirzasoleiman et al. [16] study a greedy approach to maximizing submodular set functions. They propose a distributed framework by assuming the huge data at hand is distributed among the processors, and every processor must first choose a set of candidate solutions. The central processor then applies the greedy algorithm on this restricted set, instead of the complete data set. Barbosa et al. [6] enhance this approach by randomizing the process of data distribution to the processors, and generalizing the cardinality constraint in [19] to matroid, knapsack, and p-system constraints. Singh and O'Keefe [22] work on a different area in distributed discrete optimization. They consider a scheduling problem, and solve its Lagrangean relaxation in a decentralized way.

Bertsekas [2] provides a comprehensive tutorial for auction algorithms for network problems. Although these auction algorithms are not necessarily restricted to be distributed, most of his algorithms are easily distributable. His idea of an auction is similar to the decentralized algorithms that we propose, however our underlying problem structure differs greatly from his. In Bertsekas' auction algorithm setting, the players all have a common constraint set. Their decisions are highly dependent on each other's, which is not the case in our setting. In our problems, once the resource is distributed among the players, their corresponding problems become completely independent.

In Section 2, we introduce two distributed optimization algorithms on a simplified version of the problem (P1). We generalize this simplified problem in Section 3, and provide sufficient conditions for the optimality of the algorithms described in Section 2. Section 4 includes results on approximation algorithms, i.e. distributed settings without an optimality guarantee. We finish with concluding remarks in Section 5.

2 Distributed Problems with a Cardinality Constraint

We are given a cardinality knapsack problem (P2), in which the items are distributed among m players, and each player is assigned N_i items that they can choose from. The decision variable representing whether or not player i chooses their j^{th} item is x_{ij}, i.e. the j^{th} component of the vector x_i, and the value of that decision is w_{ij}, i.e. the j^{th} component of the vector w_i. The players coordinate with each other in order to select K items in total.

$$z = \max \left\{ \sum_{i=1}^{m} w_i^T x_i : \sum_{i=1}^{m} \mathbf{1}^T x_i \le K,\ x_i \in \{0,1\}^{N_i} \ \ i = 1, \ldots, m \right\}. \tag{P2}$$

Once the resource level K is distributed among the players, such that player i gets K_i units of resource, each player gets their individual problem (P2$_i$):

$$z_i(K_i) = \max \left\{ w_i^T x_i : \mathbf{1}^T x_i \le K_i\ ,\ x_i \in \{0,1\}^{N_i} \right\}. \tag{P2$_i$}$$

Without loss of generality, we assume the indices of x_i are in decreasing order of their objective function coefficients, i.e. $w_{ij} \ge w_{i,j+1}$ for $j = 1 \ldots N_i - 1$. Thus the optimal solution x_i^* of (P2$_i$) has $x_{ij}^* = 1$ for $j = 1, \ldots, K_i$ and $x_{ij}^* = 0$ for $j = K_i + 1, \ldots, N_i$.

2.1 CC-Lin Algorithm

The first decentralized algorithm we propose for this setting works as follows: Each player is given an initial resource level K_i that satisfies $\sum_{i=1}^{m} K_i = K$. They share with each other how much their objective will increase if their resource is increased by one unit, Δ_i^+, which corresponds to w_{i,K_i+1}, and how much their objective will decrease if their resource is decreased by one unit, Δ_i^-, which corresponds to w_{i,K_i}. The player with the highest increase (i^+) and the player with the smallest decrease (i^-) swap a single unit of resource, i.e. K_{i^+} becomes $K_{i^+} + 1$ and K_{i^-} becomes $K_{i^-} - 1$. In our convergence analysis, we prove that the players i^+ and i^- are distinct unless it is the last iteration. After each iteration, all players except for i^+ and i^- remain at the same resource level, and only i^+ and i^- with the new resource levels need to update and share their Δ_i^+ and Δ_i^- values. The swapping of the resources continue until the highest increase in the objective value is not larger than the smallest decrease, and at that point the algorithm terminates. We refer to this decentralized algorithm as CC-Lin, short for *cardinality constraint linear convergence*, and describe it in Algorithm 1.

Algorithm 1 CC-Lin

```
Input: Player index $i \in \{1 \ldots m\}$, initial resource allocation $K_i$
Output: Resource allocation $K_i$
 1: Initialize: isChanged == true
 2: while true do
 3:   if isChanged== true then
 4:     isChanged= false
 5:     $\Delta_i^+ = z_i(K_i + 1) - z_i(K_i) + \epsilon(i)$; $\Delta_i^- = z_i(K_i) - z_i(K_i - 1) + \epsilon(i)$
 6:     BROADCAST($\Delta_i^+, \Delta_i^-$)
 7:   end if
 8:   RECEIVE$\left(\left\{\Delta_p^+, \Delta_p^-\right\}_{p \subseteq \{1,\ldots,m\}}\right)$
 9:   $\Delta^+ = \max_{p \in \{1 \ldots m\}} \left\{\Delta_p^+\right\}$; $i^+ = arg\max_{p \in \{1 \ldots m\}} \left\{\Delta_p^+\right\}$
10:   $\Delta^- = \min_{p \in \{1 \ldots m\}} \left\{\Delta_p^-\right\}$; $i^- = arg\min_{p \in \{1 \ldots m\}} \left\{\Delta_p^-\right\}$
11:   if $\Delta^- \ge \Delta^+$ then
12:     return $K_i$
13:   end if
14:   if $i^+ == i$ then
15:     $K_i = K_i + 1$; isChanged= true
16:   else if $i^- == i$ then
17:     $K_i = K_i - 1$; isChanged = true
18:   end if
19: end while
```

2.1.1 Convergence and Optimality

In Line 5, as a tie-breaking rule the players add a function of their indices, $\epsilon(i)$, to the Δ_i^+ and Δ_i^- values. An example for this function might be $\epsilon(i) = \varepsilon i$ for some $\varepsilon \approx 0$. $\epsilon(i)$ is significantly smaller in order of magnitude compared to Δ_i^+ and Δ_i^-, therefore it will not be included in the convergence and optimality analyses for the sake of simplicity.

Our main result on the performance of the CC-Lin algorithm is stated in Theorem 1.

Theorem 1. *Algorithm 1 applied to problem (P2) outputs an optimal resource allocation in at most K iterations.*

The following results are used to build up to the proof of Theorem 1.

Lemma 1. *When Algorithm 1 is applied to problem (P2), the players i^+ and i^- are distinct unless it is the last iteration.*

Proof. Assume $i = i^+ = i^-$, with resource value K_i. $\Delta_i^+ = w_{i,K_i+1} = \Delta^+$ and $\Delta_i^- = w_{i,K_i} = \Delta^-$ hold by our assumption. As $w_{i,K_i+1} \leq w_{i,K_i}$, we have $\Delta^+ \leq \Delta^-$, which implies that the algorithm terminates at this iteration. □

For the following proofs, let the superscript (t) refer to the value of a parameter at iteration t of the algorithm.

Lemma 2. *When Algorithm 1 is applied to problem (P2), the sequence of Δ^+ values throughout the iterations are nonincreasing, and the sequence of Δ^- values are nondecreasing.*

Proof. We will prove the first part of Lemma 2 by showing that $\Delta^{+(t)} \geq \Delta^{+(t+1)}$ for all t values. The second part, $\Delta^{-(t)} \leq \Delta^{-(t+1)}$ follows from a similar argument. Note that

$$\Delta^{+(t)} = \max_{i=1,\ldots,m} \left\{ \Delta_i^{+(t)} \right\} = \Delta_{i^{+(t)}}^{+(t)}$$

holds by the definition of Δ^+ and i^+, and this value equals $w_{i^{+(t)},K_{i^{+(t)}}+1}$, the value of the next best item player $i^{+(t)}$ will pick, with respect to their current resource level $K_{i^{+(t)}}$. We can write a similar definition of $\Delta^{+(t+1)}$ and expand it as:

$$\Delta^{+(t+1)} = \max_{i=1,\ldots,m} \left\{ \Delta_i^{+(t+1)} \right\} = \max \left\{ \max_{i \neq i^{+(t)}, i^{-(t)}} \left\{ \Delta_i^{+(t)} \right\}, \Delta_{i^{-(t)}}^{-(t)}, \Delta_{i^{+(t)}}^{+(t+1)} \right\}.$$

First notice that, after iteration t, only the players $i^{+(t)}$ and $i^{-(t)}$ update their Δ_i values, therefore, except for those two players we have $\Delta_i^{+(t+1)} = \Delta_i^{+(t)}$. By the definition of Δ^+ and i^+, $\max_{i \neq i^{+(t)}, i^{-(t)}} \left\{ \Delta_i^{+(t)} \right\} \leq \Delta^{+(t)}$. The optimality conditions of player $i^{-(t)}$ imply that the value of the item that they dropped at iteration t must be equal to the value of the next best item to be selected at iteration $t+1$, hence $\Delta_{i^{-(t)}}^{+(t+1)} = \Delta_{i^{-(t)}}^{-(t)}$. $\Delta_{i^{-(t)}}^{-(t)} = \Delta^{-(t)} \leq \Delta^{+(t)}$ trivially holds, otherwise the algorithm would terminate at iteration t. Finally $\Delta_{i^{+(t)}}^{+(t+1)}$ equals the value of the $K_{i^{+(t)}} + 2^{\text{nd}}$ best item for player $i^{+(t)}$, $w_{i^{+(t)},K_{i^{+(t)}}+2}$, which is at most $w_{i^{+(t)},K_{i^{+(t)}}+1}$, or equivalently $\Delta^{+(t)}$. Thus $\Delta^{+(t+1)} \leq \Delta^{+(t)}$.

□

Lemma 3. *When Algorithm 1 is applied to problem (P2), the changes in K_i values are monotonous for every player i throughout the algorithm.*

Proof. Similar to the previous proof, we will only provide the proof for one side of the argument, namely that a player who increases their resource at some iteration cannot decrease it at a later iteration. The other direction, i.e. a player who decreases their resource at some iteration cannot increase it at a later iteration, can be proven using a similar argument.

For contradiction, let player i' be a player who has an increase of 1 unit in resource $K_{i'}$ at iteration t', and a later decrease of 1 unit at iteration t'', and $K_{i'}$ stays constant in between iterations t' and t''. We have shown in Lemma 2 that $\Delta^{+(t')} \geq \Delta^{+(t'')}$ as $t' \leq t''$. We define t' to be the iteration when i' is the player with the highest gain and t'' when i' is the player with the smallest loss, therefore we know that $\Delta^{+(t')} = \Delta_{i'}^{+(t')}$ and $\Delta^{-(t'')} = \Delta_{i'}^{-(t'')}$. Furthermore, the value of the next best item picked at iteration t' equals the value of the first item to be dropped until iteration t'', so $\Delta_{i'}^{+(t')} = \Delta_{i'}^{-(t)}$ for $t = t'+1, \dots, t''$. Using these arguments, we get the chain

$$\Delta^{-(t'')} = \Delta_{i'}^{-(t'')} = \Delta_{i'}^{+(t')} = \Delta^{+(t')} \geq \Delta^{+(t'')}. \tag{1}$$

As by our assumption, the algorithm has not terminated at iteration t'', which is only possible if $\Delta^{-(t'')} < \Delta^{+(t'')}$. Therefore (1) yields a contradiction, so there exists no such player i'.

□

Proof of Theorem 1. Let $\{\overline{K}_i\}_{i=1}^m$ be the resource allocation Algorithm 1 outputs, and $\{K_i^*\}_{i=1}^m$ be an optimal resource allocation with minimum Hamming distance to $\{\overline{K}_i\}_{i=1}^m$. As a contradiction, assume that $\{\overline{K}_i\}_{i=1}^m \neq \{K_i^*\}_{i=1}^m$, and $\sum_{i=1}^m z_i(K_i^*) > \sum_{i=1}^m z_i(\overline{K}_i)$. Denote by S^+ the set of players i with $K_i^* > \overline{K}_i$, and S^- the set of players i with $\overline{K}_i > K_i^*$. We have

$$\begin{aligned}
\sum_{i=1}^m z_i(K_i^*) - \sum_{i=1}^m z_i(\overline{K}_i) &= \sum_{i\in S^+} \big(z_i(K_i^*) - z_i(\overline{K}_i)\big) - \sum_{i\in S^-} \big(z_i(\overline{K}_i) - z_i(K_i^*)\big) && (2)\\
&= \sum_{i\in S^+} \sum_{j=\overline{K}_i+1}^{K_i^*} w_{ij} - \sum_{i\in S^-} \sum_{j=K_i^*+1}^{\overline{K}_i} w_{ij} && (3)\\
&\leq \sum_{i\in S^+} w_{i,\overline{K}_i+1}(K_i^* - \overline{K}_i) - \sum_{i\in S^-} w_{i,\overline{K}_i}(\overline{K}_i - K_i^*) && (4)\\
&= \sum_{i\in S^+} \Delta_i^+(K_i^* - \overline{K}_i) - \sum_{i\in S^-} \Delta_i^-(\overline{K}_i - K_i^*) && (5)\\
&\leq (\Delta^+ - \Delta^-) \sum_{i\in S^+} (K_i^* - \overline{K}_i) && (6)\\
&\leq 0.
\end{aligned}$$

We partition the sum into two sets to get (2). The marginal changes $z_i(K_i+1) - z_i(K_i)$ correspond to the weight w_{i,K_i+1}, and this substitution gives us (3). The order of the items imply $w_{i,K_i} \geq w_{i,K_i+1}$, and we use these bounds to get (4). According to the distribution $\{\overline{K}_i\}_{i=1}^m$, $w_{i,\overline{K}_i+1}$ is the weight of the first item to be picked by player i, which is Δ_i^+, $w_{i,\overline{K}_i}$ is the weight of the first item to be dropped off, Δ_i^-, hence (5). Δ_i^+ is trivially upperbounded by Δ^+ and Δ_i^- is lowerbounded by Δ^-. Taking those two out of the summation, and the fact that $\sum_{i\in S^+}(K_i^* - \overline{K}_i) = \sum_{i\in S^-}(\overline{K}_i - K_i^*)$ yields (6), and since $\Delta^+ < \Delta^-$ at termination, this term is less than or equal to zero. This conclusion contradicts our assumption that $\sum_{i=1}^m z_i(K_i^*) > \sum_{i=1}^m z_i(\overline{K}_i)$, therefore no such $\{K_i^*\}_{i=1}^m$ exists, $\{\overline{K}_i\}_{i=1}^m$ is the optimal allocation.

Let $\{K_i'\}_{i=1}^m$ be the resource allocation used to initialize Algorithm 1. Consider the set $S^+ = \{i : \overline{K}_i > K_i'\}$ and a player $i \in S^+$. By the construction of our algorithm, K_i increases by 1 unit only when player i is the player with the highest gain and is i^+. Lemma 3 implies that player i is i^+ at exactly $\overline{K}_i - K_i'$ iterations. The algorithm allows only 1 winner at each iteration, therefore it requires $\sum_{i \in S^+} \overline{K}_i - K_i'$ iterations for players in $i \in S^+$ to reach resource level $\overline{K}_i$. A similar argument can be made for players in $S^- = \{i : \overline{K}_i < K_i'\}$, and it requires $\sum_{i \in S^-} K_i' - \overline{K}_i$ iterations for them to reach $\overline{K}_i$. Finally, the players in neither of these sets cannot be i^+ or i^- at any iteration, by Lemma 3. Notice that the two processes, selecting i^+ or i^-, are performed concurrently, therefore the algorithm terminates in $\sum_{i \in S^-} K_i' - \overline{K}_i = \sum_{i \in S^+} \overline{K}_i - K_i' \leq K$ iterations. $\square$

2.1.2 Communication Complexity

For the problem to be solved in a distributed manner, the players need to share two pieces of information with each other: how much an additional unit of resource is worth to them (Δ_i^+) and how much one less unit of resource will cost them(Δ_i^-). The function $BROADCAST()$ is used to communicate this information. In the first iteration, $BROADCAST()$ is called m times (once for each player), where all processors broadcast to all processors. This communication scheme is called *Multinode broadcast.* In the remaining iterations $BROADCAST()$ is only called twice (only by players i^+ and i^-). In that case the type of communication is called *Single node accumulation.* Optimal communication times for multinode broadcast and single node accumulation in various network structures are given in Table 2.1.2.

	Ring	Tree	Mesh
Single node accumulation	$\Theta(m)$	$\Theta(\log m)$	$\Theta(m^{1/d})$
Multinode broadcast	$\Theta(m)$	$\Theta(m)$	$\Theta(m)$

Table 1: Solution times of optimal algorithms for the basic communication problems using a ring, a binary balanced tree, and a d-dimensional symmetric mesh with m processors [3]

CC-Lin in its current description exploits the memory of processors in order to minimize communication by storing the Δ_i^+ and Δ_i^- values of each player. Other communication designs are also possible. Given a communication topology in the underlying network, in two rounds of communication the same goal can be achieved. Namely in the first round Δ_i^+, i^+, Δ_i^-, i^- are computed, and in the second round this information is broadcasted to all players. The advantage of this communication scheme is the lack of memory requirement, and the disadvantage is that every player has to share and receive data at all iterations.

2.2 CC-Log Algorithm

The next algorithm that we propose is based on the distributed algorithm for finding the median in [21]. Consider the setting in (P2) with only two players, $m = 2$, where without loss of generality each player has K items with corresponding item values, i.e. $N_i = K$, and the task is to find the median of the item values. Given an array U of size n of non-increasing values, let $median1(U)$ be the $\lfloor \frac{n}{2} \rfloor^{th}$ element. Notice that for all optimal solutions x^* of (P2), if $x_{ij}^* = 1$, then $w_{ij}^* \geq median1(w)$ and if $x_{ij}^* = 0$, then $w_{ij}^* \leq median1(w)$. In other words, for an optimal allocation $\{K_1^*, K_2^*\}$, $median1(w) = \min\{w_{1,K_1^*}, w_{2,K_2^*}\}$.

Bearing in mind the similarities between our problem (P2) and the problem of finding the median, we propose the following variant of the algorithm in [21]. Players 1 and 2 both start with zero initial resource allocation, i.e. $K_i = 0$, and the remaining resource, $\overline{K}$, is initially K. They share one piece of information with each other, Δ_i: assuming they have half of the remaining resource, what would be the value of the next best item they would select. Namely $\Delta_i = z_i(K_i + \lfloor \frac{\overline{K}}{2} \rfloor + 1) - z_i(K_i + \lfloor \frac{\overline{K}}{2} \rfloor)$. This value can be interpreted as the median of the values of the unselected items of player i. Let $i^+ = argmax\{\Delta_i\}$. Player i^+ gets half of the remaining resource, and the remaining resource is reduced by half. This process is repeated until the remaining resource is zero. We refer to this decentralized algorithm as CC-Log, short for *cardinality constraint logarithmic convergence*, and describe it in Algorithm 2.

Algorithm 2 CC-Log

```
Input: Player index $i$, total resource $K$
Output: Resource allocation $K_i$
1: Initialize: Remaining resource $\overline{K} = K$, allocated resource $K_i = 0$
2: while $\overline{K} > 0$ do
3:     $\Delta_i = z_i(K_i + \lfloor \frac{\overline{K}}{2} \rfloor + 1) - z_i(K_i + \lfloor \frac{\overline{K}}{2} \rfloor) + \epsilon(i)$
4:     BROADCAST($\Delta_i$)
5:     RECEIVE($\Delta_{i'}$)
6:     if $\Delta_i > \Delta_{i'}$ then
7:         $K_i = K_i + \lceil \frac{\overline{K}}{2} \rceil$
8:     end if
9:     $\overline{K} = \overline{K} - \lceil \frac{\overline{K}}{2} \rceil$
10: end while
```

CC-Log is shown to terminate in $\log K$ iterations [21], as at each iteration $\overline{K}$ is reduced by half. The optimality for the two player setting comes from an inductive proof using Theorem 2.

Theorem 2. *Let $\Delta_i = z_i(\lfloor \frac{\overline{K}}{2} \rfloor + 1) - z_i(\lfloor \frac{\overline{K}}{2} \rfloor)$ for $i = 1, 2$. If $\Delta_1 \geq \Delta_2$, then there exists an optimal resouce allocation $\{K_1^*, K_2^*\}$ that satisfies $K_1^* \geq \lceil \frac{\overline{K}}{2} \rceil$.*

Proof. Assume otherwise. Namely, for all optimal resource allocations assume $K_1 < \lceil \frac{\overline{K}}{2} \rceil$ holds. Let $\{K_1^*, K_2^*\}$ be an optimal resource allocation with maximum K_1^*. Define $z^* = z_1(K_1^*) + z_2(K_2^*)$ and $\bar{z} = z_1(K_1^* + 1) + z_2(K_2^* - 1)$. We have

$$\begin{aligned}
\bar{z} &= z^* + z_1(K_1^* + 1) - z_1(K_1^*) + z_2(K_2^*) - z_2(K_2^* - 1) \\
&\geq z^* + \Delta_1 - \Delta_2 \\
&\geq z^*.
\end{aligned}$$

Notice that since $K_1^* < \lceil \frac{\overline{K}}{2} \rceil$, the value of the $(K_1^* + 1)^{st}$ item of player 1, $z_1(K_1^* + 1) - z_1(K_1^*)$, is at least the value of the $(\lfloor \frac{\overline{K}}{2} \rfloor + 1)^{th}$ item, Δ_1. Similarly, the value of the K_2^{*th} item of player 2, $z_2(K_2^*) - z_2(K_2^* - 1)$, is at most Δ_2. By our assumption we have $\Delta_1 \geq \Delta_2$, therefore $\bar{z} \geq z^*$ is implied. Either $\bar{z} > z^*$ contradicting the optimality of $\{K_1^*, K_2^*\}$, or $\bar{z} = z^*$ meaning $\{K_1^* + 1, K_2^* - 1\}$ is an optimal solution with higher resource allocated to player 1, contradicting our assumption. □

Theorem 3. *Algorithm 2 outputs an optimal resource allocation for problem (P2) for $m = 2$.*

Proof. Each iteration of Algorithm 2 can be interpreted as discarding half of the feasible region of resource allocations, until we get a single point as the remaining feasible region. Theorem 2 indicates that there exists at least one optimal solution in the remaining feasible region after each iteration. Therefore, inductively we can conclude that the last feasible solution remaining is an optimal solution. □

We generalize Algorithm 2 to a multiplayer setting by partitioning the players into two groups, having the union of the players in a group act as a single entity, and thereby imitate the two player game. The new algorithm, also called CC-Log, is described in Algorithm 3.

It can easily be seen that Algorithm 3 is structually equivalent to Algorithm 2. The main challange of Algorithm 3 is Line 3, where

Algorithm 3 CC-Log

```
Input: Set of players $I \subseteq \{1 \dots m\}$, total resource $K$
Output: Resource allocation $K_I$
 1: Initialize: Remaining resource $\overline{K} = K$, allocated resource $K_I = 0$
 2: while $\overline{K} > 0$ do
 3:     $\Delta_I = z_I(K_I + \lfloor \frac{\overline{K}}{2} \rfloor + 1) - z_I(K_I + \lfloor \frac{\overline{K}}{2} \rfloor) + \epsilon(I)$
 4:     BROADCAST($\Delta_I$)
 5:     RECEIVE($\Delta_{\bar{I}}$)
 6:     if $\Delta_I > \Delta_{\bar{I}}$ then
 7:         $K_I = K_I + \lceil \frac{\overline{K}}{2} \rceil$
 8:     end if
 9:     $\overline{K} = \overline{K} - \lceil \frac{\overline{K}}{2} \rceil$
10: end while
```

$$z_I(K_I) = \max \left\{ \sum_{i \in I} w_i^T x_i : \sum_{i \in I} \mathbf{1}^T x_i \le K_I,\ x_i \in \{0,1\}^{N_i} \ \ i \in I \right\}. \tag{P_I}$$

In other words, $z_I(K_I)$ is the resulting objective value when the set of players I allocate K_I units of resource optimally among each other. Notice that our main problem (P2) is a special case of (P_I) where $I = \{1, \dots, m\}$ and $K_I = K$. Therefore solving (P_I) is at least as hard as solving the original problem (P2). We propose to solve (P_I) using the CC-Log algorithm, Algorithm 3. At every iteration of CC-Log(I), in order to compute $z_I(\cdot)$ the group of players I run the algorithm CC-Log(I') against CC-Log($I \backslash I'$) for some $\emptyset \subset I' \subset I$. Unless $|I| = 2$, at least one of the sets I' and $I \backslash I'$ has size strictly greater than one, and hence CC-Log called for that set is Algorithm 3 again. The recursion stops when the set is a singleton and the CC-Log function called is Algorithm 2.

2.2.1 Convergence and Optimality

In order to analyze the convergence of the algorithm, we will construct the following game tree. Every node v has a non-empty group of players associated with it, $I_v \subseteq \{1, \dots, m\}$. For the root node r, we have $I_r = \{1, \dots, m\}$. A node v has two child nodes v_1 and v_2, unless $|I_v| = 1$, in which case v is a leaf node. The set of players associated with v is partitioned to v_1 and v_2, i.e. $I_v = I_{v_1} \cup I_{v_2}$ and $I_{v_1} \cap I_{v_2} = \emptyset$. We use this game tree to model how the set of players is partitioned into groups to compute $z_I(\cdot)$ and run the decentralized algorithm CC-Log. Our assumption is that the two siblings v_1 and v_2 run CC-Log against each other to compute $z_{I_v}(\cdot)$ for their parent v. Note that all nodes except the root node have one sibling and one parent.

Given such a game tree, the two children of the root node r start by playing the decentralized game against each other, where we have CC-Log(I_{r_1}) against CC-Log(I_{r_2}). At every iteration of the algorithm CC-Log(I_v), in order to compute $z_{I_v}(\cdot)$, node v starts a new game between the two children CC-Log(I_{v_1}) against CC-Log(I_{v_2}), unless $|I_v| = 1$. Let $\gamma(v, K_{I_v})$ be the number of iterations CC-Log requires to find $z_{I_v}(K_{I_v})$. In other words $\gamma(v, K_{I_v})$ is the number of iterations required for the game CC-Log(I_{v_1}) against CC-Log(I_{v_2}) to converge with initial remaining resource $\overline{K} = K_{I_v}$, where v_1 and v_2 are the children of v. In the case where $|I_v| = 1$ and v_1 and v_2 don't exist, we assume $\gamma(v, K_{I_v}) = 1$. Recall that as explained in [21], the while loop in CC-Log takes $\log K_{I_v}$ iterations, and at every iteration we call CC-Log for both of the child nodes with $\overline{K}$ values at most K_{I_v}. Thus we have the recursion:

$$\gamma(v, K_{I_v}) \le \log K_{I_v} \cdot \max \{\gamma(v_1, K_{I_v}), \gamma(v_2, K_{I_v})\}. \tag{7}$$

Theorem 4. *Algorithm 3 applied to (P2) terminates in at most* $(\log K)^{\alpha}$ *iterations, where* α *varies between* $\log m$ *and* m *depending on the communication network structure.*

Proof. The value we need to bound to prove Theorem 4 is $\gamma(r, K)$. Notice that the bound (7) multiplies by a factor of $\log K$ with each level of children, until we reach a node with both children as leaf nodes in which case $\max\{\gamma(v_1, K), \gamma(v_2, K)\} = 1$. Therefore, the bound (7) for $\gamma(r, K)$ becomes $(\log K)^{\alpha}$, where α is the depth of the game tree. The structure of the tree highly depends on the communication network between the players, i.e. which players can communicate directly with each other and can act as a single group in Algorithm 3. For a perfectly balanced game tree as the best case scenario, i.e. a tree with leaf nodes having almost the same depth, we have $\alpha = \log m$. For a skewed tree as the worst case scenario, i.e. a tree with almost all leaf nodes having different depths, we have $\alpha = m - 1$.

$\square$

For the rest of the paper, we will assume a complete communication network, and a balanced game tree, and hence a $(\log K)^{\log m}$ upperbound on the convergence of the CC-Log algorithm. Although the impact of K on the convergence has been reduced to $\log K$, now the convergence bound also involves m. Thus neither the CC-Lin or the CC-Log algorithm dominates the other.

Theorem 5. *Algorithm 3 applied to (P2) outputs an optimal resource allocation.*

Proof. Recall that given a game tree, for a node v our algorithm solves for $z_{I_v}(\cdot)$ by having the two child nodes v_1 and v_2 play CC-Log against each other. The main goal is to show the optimality of $z_{I_r}(K)$, which we will prove by induction on $|I_v|$. For a node v with $|I_v| = 1$, $z_{I_v}(\cdot)$ is computed optimally as it is the player's individual problem (P2$_i$) for some player i.

Now assume $|I_v| = \alpha$ for some $\alpha > 2$ and that $z_I(\cdot)$ is computed optimally for all sets I with $|I| < \alpha$. In order to compute $z_{I_v}(\cdot)$, the child nodes play CC-Log(I_{v_1}) against CC-Log(I_{v_2}). Let i_1 and i_2 be two dummy players who have complete information on the problems of players I_{v_1} and I_{v_2} respectively, and define $z_{i_j}(\cdot)$ to be the optimal solution of the problem (P_I) for $I = I_{v_j}$ for $j = 1, 2$. By our induction, since $|I_{v_j}| < |I_v| = \alpha$ we assume that $z_{I_{v_j}}(\cdot)$ computed by multiplayer CC-Log is optimal, i.e. $z_{I_{v_j}}(\cdot)$ used in Algorithm 3 by v_j is equal to $z_{i_j}(\cdot)$ for $j = 1, 2$. The information shared in the two player Algorithm 2 between i_1 and i_2 is identical to the information shared in the multiplayer Algorithm 3 between I_{v_1} and I_{v_2}. Therefore the resource allocation decisions at each iteration and the output of the two player Algorithm 2 between i_1 and i_2 equal resource allocation decisions at each iteration and the output of the multiplayer Algorithm 3 between I_{v_1} and I_{v_2}. Thus, the optimality guarantee of the two player game in Theorem 3 holds for the multiplayer game, and hence z_{I_v} is computed optimally. $\square$

3 Generalizing the Problem Structure

3.0.1 Accommodating Local Constraints

The decentralized algorithms CC-Lin and CC-Log are introduced and analyzed on a very simple item selection problem, however, their applications are not restricted to it. In this section we first provide a sufficient condition of optimality for more general problem types and then we give examples of problems that satisfy this sufficient condition.

Given the general problem (P1) and the players' individual problems (P1$_i$), construct the problems:

$$\widetilde{z} = \max\left\{\sum_{i=1}^{m} \widetilde{w}_i^T y_i \;:\; \sum_{i=1}^{m} \mathbf{1}^T y_i \leq K,\; y_i \in \{0,1\}^{N_i}\;\; i = 1, \ldots, m\right\} \tag{P3}$$

$$\widetilde{z}_i(K_i) = \max\left\{\widetilde{w}_i^T y_i \ : \ \mathbf{1}^T y_i \leq K_i, \ y_i \in \{0,1\}^{N_i}\right\} \tag{P3$_i$}$$

where $N_i = K$ for all players $i = 1 \dots m$, and the weights $\widetilde{w}_{ij} = z_i(j) - z_i(j-1)$ for $j = 1 \dots K$. In other words, we treat the marginal changes in the objective function value with each additional unit of resource as an item, where the value of the item equals the value of the marginal increase. Notice that (P3) is structually equivalent to (P2), and therefore has the convergence properties discussed in Section 2.

Definition 1 (Concavity Property)**.** The problem (P1$_i$) is defined to have the *concavity property* when the marginal changes in the objective function value do not increase as K_i increases, i.e. problem (P1$_i$) has the concavity property when $z_i(K_i+2) - z_i(K_i+1) \leq z_i(K_i+1) - z_i(K_i)$ for all $K_i \geq 0$.

The concavity property of a player's individual problem (P1$_i$) implies that the player's items in problem (P3$_i$) are ordered. Namely, $\widetilde{w}_{i,j} \geq \widetilde{w}_{i,j+1}$. This observation leads us to the conclusion that $z_i(K_i) = \widetilde{z}_i(K_i)$ for all integer $K_i = 0, \dots, K$ because

$$\begin{aligned}
\widetilde{z}_i(K_i) &= \sum_{j=1}^{K_i} \widetilde{w}_{ij} && (8)\\
&= \sum_{j=1}^{K_i} z_i(j) - z_i(j-1) && (9)\\
&= z_i(K_i). && (10)
\end{aligned}$$

Here, (8) is the optimal solution of the cardinality knapsack problem (P3$_i$), as the K_i most valuable items are items $j = 1, \dots, K_i$ by the concavity property. (9) is by the definition of $\widetilde{w}_{ij}$, and we get (10) by simplification. Furthermore, due to our initial assumption of all data being integer, the objective function $z_i(K_i)$ of (P1$_i$) is a discrete function of K_i. Namely, for any $\epsilon \in [0,1)$ and $K_i \in \mathbb{Z}_+$, $z_i(K_i) = z_i(K_i + \epsilon)$. With this observation, it suffices to show the equivalence of z_i and $\widetilde{z}_i$ for integer K_i values. When the objective functions behave identically, the information each player shares, Δ_i^+ and Δ_i^- in CC-Lin and Δ_i in CC-Log, is the same whether they are solving problem (P1$_i$) or (P3$_i$), which implies that the reassignment of the resources is also the same. Hence, when problems (P3$_i$) have the concavity property for all players $i = 1, \dots, m$, the steps of CC-Lin and CC-Log when solving (P1) are the same as when solving (P3). All the convergence properties that hold for (P3$_i$), i.e. a single cardinality knapsack problem as discussed in Sect. 2, also holds for (P1) when (P1$_i$) has the concavity property for all players $i = 1, \dots, m$. Namely,CC-Lin outputs the optimal solution in at most K iterations, and CC-Log outputs the optimal solution in at most $(\log K)^{\log m}$ iterations.

3.0.2 Example Problems with the Concavity Property

In order to prove concavity, we restrict ourselves to problems (P1) that can be reformulated in the following structure:

$$z = \max\left\{\sum_{i=1}^{m} w_i^T x_i : \sum_{i=1}^{m} \mathbf{1}^T x_i \leq K \ , \ x_i \in X_i \ , \ x_i \in \{0,1\}^{N_i} \quad i = 1, \dots, m\right\}. \tag{P4}$$

Along with the cardinality constraint, each player has another set of constraint it needs to satisfy, which are described by $x_i \in X_i$. When we distribute (P4) into each player's individual subproblems, we get

$$z_i(K_i) = \max\left\{ w_i^T x_i : \mathbf{1}^T x_i \le K_i \ , \ x_i \in X_i \ , \ x_i \in \{0,1\}^{N_i} \right\}. \tag{P4$_i$}$$

Suppose that the convex hull of X_i, $conv(X_i)$ is integral, i.e. all of its extreme points are integer. Aghezzaf and Wolsey [1] showed that if $conv\left(X_i \cap \left\{\sum_{j=1}^{N_i} x_{ij} = k\right\}\right)$ is integral for all nonnegative integer k values, then (P4$_i$) has the concavity property. It is known that matroid, matching and matroid intersection polytopes are integral when intersected with a cardinality constraint [18], and thus possess the concavity property. By combining the definitions of binary and integer b-matching polytopes in [18] and the integrality result of the matching polytope, we conclude that the binary and integer b-matching problems also have the concavity property. Furthermore, Walter et.al. [23] show that transportation problem with market choice satisfies this condition. Thus for any of these problems with a cardinality constraint, all of the results of Section 2 are valid.

3.0.3 Distributed Problems with a Cardinality Objective

Consider the problem (P5)

$$z = \max\left\{ \sum_{i=1}^{m} \mathbf{1}^T x_i : \sum_{i=1}^{m} a_i^T x_i \le K, \ x_i \in X_i \ i = 1, \ldots, m \right\}. \tag{P5}$$

It is a knapsack problem with arbitrary item sizes, and the objective is maximizing the number of items selected. Again, the items are distributed among m players, and the i^{th} player is assigned N_i items to choose from, where $\sum_i N_i = N$. The decision variable representing whether or not player i chooses its j^{th} item is $x_{ij} \in \{0,1\}$, i.e. the j^{th} component of x_i, the size of that item is a_{ij}, i.e. the j^{th} component of a_i, and X_i is the set of local constraints for player i. The players are expected to coordinate with each other in order to select the maximum number of items and not exceed the total capacity K.

There exists distributed algorithms for problem (P5) that have an optimality guarantee and are bound to terminate in $O(N)$ iterations for all polytopes X_i mentioned in Section 3.0.2 [11].

4 Approximation Algorithms

We assume the same resource allocation problem (P1) in the distributed setting, where the individual players solve (P1$_i$). The sufficient optimality condition for the decentralized CC-Lin and CC-Log algorithms in Section 2 only holds for problems (P1$_i$) with the concavity property. For problems that do not have the concavity property, one option is to relax the optimality condition and consider approximations. In this section, we focus on heuristics with provable bounds for problems without the concavity property. We also test the performance of the CC-Lin algorithm on non-concave functions to gain some insight on the average case performance.

4.1 Using Concave Approximations

Consider problems (P1$_i$) that do not have the concavity property and for which there exist concave approximation functions. Our proposed approach is to use these concave approximation functions during the decentralized algorithms. By doing this, we give up on the optimality guarantee for our original problem, but exploit the fact that the algorithms have an optimality guarantee with respect to the concave approximation functions used. An analysis of the decentralized algorithms' accuracy when concave approximation functions are used is provided in this section. Recall that when the function is concave, CC-Lin and

CC-Log output the same resource allocation, i.e. the optimal allocation with respect to the concave function. Therefore, the accuracy analyses and error bounds will be the same for both of the algorithms.

4.1.1 Error bounds

Let $\overline{z}_i$ be an approximation of z_i that has the concavity property, and satisfies at least one of

$$z_i(K_i) - \mu_i^L \leq \quad \overline{z}_i(K_i) \quad \leq z_i(K_i) + \mu_i^U \tag{11}$$
$$\rho_i^L z_i(K_i) \leq \quad \overline{z}_i(K_i) \quad \leq \rho_i^U z_i(K_i) \tag{12}$$

for $\mu_i^L, \mu_i^U \geq 0$, $\rho_i^L \leq 1$ and $\rho_i^U \geq 1$. Concave relaxations of z_i will have $\mu_i^L = 0$ and $\rho_i^L = 1$, and concave heuristics will have $\mu_i^U = 0$ and $\rho_i^U = 1$. However $\overline{z}_i$ can be any arbitrary concave function that satisfies the bounds on $\mu_i^L, \mu_i^U, \rho_i^L$ and ρ_i^U. It is also possible to model noise in the data or error caused by communication lags in this form.

First, let us consider the case when the $\overline{z}_i$s satistfy (11). Let $\{K_i^*\}_{i=1}^m$ be the optimal allocation of the resource and $\{\overline{K}_i\}_{i=1}^m$ be the allocation decentralized algorithms output when using $\overline{z}_i$. We define $S^+ = \{i \mid K_i^* > \overline{K}_i\}$ as the set of players who have more resource in the optimal allocation than in the output of the algorithm, $S^- = \{i \mid K_i^* < \overline{K}_i\}$ as the set of players who have less resource in the output of the algorithm than the optimal allocation. Then we have

$$\begin{aligned}
\sum_{i=1}^m z_i(K_i^*) - \sum_{i=1}^m z_i(\overline{K}_i) &= \sum_{i\in S^+} \big(z_i(K_i^*) - z_i(\overline{K}_i)\big) - \sum_{i\in S^-} \big(z_i(\overline{K}_i) - z_i(K_i^*)\big) && (13)\\
&\leq \sum_{i\in S^+} \big(\overline{z}_i(K_i^*) + \mu_i^L - \overline{z}_i(\overline{K}_i) + \mu_i^U\big) \\
&\quad - \sum_{i\in S^-} \big(\overline{z}_i(\overline{K}_i) - \mu_i^U - \overline{z}_i(K_i^*) - \mu_i^L\big) && (14)\\
&\leq \sum_{i\in S^+} \Delta_i^+(K_i^* - \overline{K}_i) - \sum_{i\in S^-} \Delta_i^-(\overline{K}_i - K_i^*) \\
&\quad + \sum_{i\in S^+\cup S^-} \mu_i^L + \mu_i^U && (15)\\
&= (\Delta^+ - \Delta^-) \sum_{i\in S^+} (K_i^* - \overline{K}_i) + \sum_{i\in S^+\cup S^-} \mu_i^L + \mu_i^U && (16)\\
&\leq \sum_{i\in S^+\cup S^-} \mu_i^L + \mu_i^U && (17)\\
&\leq \max_{i=1,\ldots,m} \left\{\mu_i^L + \mu_i^U\right\} \cdot \min\{m, K\}. && (18)
\end{aligned}$$

Having (13), we replace z_i with appropriate bounds of $\overline{z}_i$ to get (14). Knowing that $\overline{z}_i$ are concave functions of K, Δ_i^+ is an upperbound on $\overline{z}_i(K+1) - \overline{z}_i(K)$ for $K \geq \overline{K}_i$ and $i \in S^+$, and Δ_i^- is a lowerbound on $\overline{z}_i(K) - \overline{z}_i(K-1)$ for $K \leq \overline{K}_i$ and $i \in S^-$, hence (15). The definition of Δ^+ and Δ^- together with the fact that $\sum_{i\in S^+}(K_i^* - \overline{K}_i) = \sum_{i\in S^+}(K_i^* - \overline{K}_i) \geq 0$ imply (16), and $\Delta^+ \leq \Delta^-$ at termination implies (17). Finally, the natural upperbound on $|S_i^+ \cup S_i^-|$ gives us (18).

The analysis for $\overline{z}_i$ when it satisfies (12) is quite straightforward, i.e.

$$\frac{\sum_{i=1}^m z_i(K_i^*)}{\sum_{i=1}^m z_i(\overline{K}_i)} \leq \frac{\sum_{i=1}^m \frac{1}{\rho_i^L}\overline{z}_i(K_i^*)}{\sum_{i=1}^m \frac{1}{\rho_i^U}\overline{z}_i(\overline{K}_i)} \tag{19}$$

$$\leq \frac{\frac{1}{\min_i \rho_i^L}\sum_{i=1}^m \overline{z}_i(K_i^*)}{\frac{1}{\max_i \rho_i^U}\sum_{i=1}^m \overline{z}_i(\overline{K}_i)} \tag{20}$$

$$\leq \frac{\max_i \rho_i^U}{\min_i \rho_i^L}. \tag{21}$$

Since $\overline{z}_i$ has the concavity property, the output of the decentralized algorithm is optimal for $\overline{z}_i$. This means $\sum_{i=1}^m \overline{z}_i(\overline{K}_i) \geq \sum_{i=1}^m \overline{z}_i(K_i^*)$, and therefore (19) implies (21).

In situations where the players' problems are hard, player i may use a heuristic $z_i^H(K_i)$. In this case, comparing the true optimal, $\sum_{i=1}^m z_i(K_i^*)$, with the realized value, $z_i^H(\overline{K}_i)$ can bring additional insight. If we assume $\overline{z}_i(K_i) \leq z_i^H(K_i) + \mu_i^{HU}$ and $\overline{z}_i(K_i) \leq \rho_i^{HU} z_i^H(K_i)$ holds for the heuristic z_i^H, then (18) and (21) can be modified by similar arguments to get

$$\sum_{i=1}^m z_i(K_i^*) - \sum_{i=1}^m z_i^H(\overline{K}_i) \leq \max_{i=1,\ldots,m}\left\{\mu_i^L + \mu_i^{HU}\right\} \cdot \min\{m, K\} \tag{22}$$

$$\frac{\sum_{i=1}^m z_i(K_i^*)}{\sum_{i=1}^m z_i^H(\overline{K}_i)} \leq \frac{\max_i \rho_i^{HU}}{\min_i \rho_i^L}. \tag{23}$$

4.1.2 Example concave approximations

A greedy algorithm, defined as choosing an item with the largest contribution to the objective function at each iteration, is a natural candidate for concave approximations. It is well known that the output of the greedy algorithm has the concavity property for submodular set functions [19]. We have shown in [11] the error bounds in the form of (12) for the problems of maximizing submodular set functions [19], maximizing a linear function over an independent system [13], and maximizing a non-decreasing submodular function over an independent system [8]. Some of these selected problems inherently have the cardinality constraint, which we require in our setting. For those that don't, we provide error bounds for the problem with the cardinality constraint included.

Another candidate for concave approximations is linear relaxations. Given $z(K) = \max\left\{c^T x : Ax \leq b,\ \mathbf{1}^T x \leq K,\ x \in \{0,1\}^n\right\}$, an optimization problem with respect to a cardinality constraint where all data is integer, $\bar{z}(K) = \max\left\{c^T x : Ax \leq b,\ \mathbf{1}^T x \leq K,\ x \in [0,1]^n\right\}$, which is the linear relaxation, is a concave function on integer K values. Clearly $\bar{z}(K) \geq z(K)$ holds, and is not always satisfied at equality. For some problems, there are results in the literature for bounding the linear relaxation from both sides in the form of (12). We have shown in [11] the error bounds for the maximum $l-$SAT problem [4] and the weighted matchoid problem [14].

4.2 Experiments on CC-Lin with arbitrary Knapsack problems

Returning back to the general problem (P1), when the individual problems (P1$_i$) do not have a structure like the concavity property, the algorithms can be shown to perform arbitrarily badly in terms of accuracy. Regardless of this theoretic lower bound, in this section we present empirical results on their average behavior. We generate random knapsack problems and apply the CC-Lin algorithm on these non-structured problems.

Notice that regardless of the structure of the problem, the stopping condition of the CC-Lin algorithm, $\Delta^- \geq \Delta^+$, is a necessary condition for optimality. In other words, at an optimal solution, there exists no improving movements. Recall that CC-Lin starts from an arbitrary resource allocation. If, by luck, the algorithm is initialized at the optimal solution, it terminates immediately, as there is no improving movement, and returns the optimal solution. This means that the performance of CC-Lin strongly depends on the initial resource allocation. Therefore, we propose a heuristic based on CC-Lin that runs the CC-Lin algorithm multiple times with different initial resource allocations each time.

In addition, in the absence of the concavity property, when applying CC-Lin to non-concave functions, we cannot guarantee that the winning player i^+ and the losing player i^- in each iteration are distinct. In fact, it is easy to construct examples with $i^+ = i^-$ when the problems don't have the concavity property. In order to avoid this kind of cycling, we modify the algorithm to terminate when $i^+ = i^-$. Although it is possible to enhance this approach by considering the second best winning and losing players when $i^+ = i^-$ to guarantee an optimal solution, this could lead to an arbitrarily long decision process.

We include one final modification to our algorithm. Let player i_1 have resource K_{i_1} which is more than she has need for, i.e. $K_{i_1} > \sum_{j=1}^{N_{i_1}} a_{i_1 j}$. In this case, the value of one less unit of resource is zero for this player, $\Delta_{i_1}^- = 0$. Now consider the remaining players, and assume that one unit of resource will not change their objective function values, and that the next jump occurs at, say, two extra units of resource. As it is, their Δ_i^+ values are 0, and the algorithm does not allow the unused resource of player i_1 to move to other players, where it might potentially increase the objective function values in the upcoming iterations. In order to avoid this, we make the unused resource more favorable to be transferred by setting $\Delta_i^- = -\epsilon$ for players i that satisfy $K_i > \sum_{j=1}^{N_i} a_{ij}$.

In our heuristic, we run CC-Lin $m + 3$ times, where m is the number of players, using the following initialization methods:

1. **ExtrPt:** We initialize the algorithm m times, once for every extreme point of the polytope $\{\sum_{i=1}^m K_i = K, K_i \geq 0\}$, i.e. one player gets all the resource. This initialization method outputs the highest value obtained from these m initializations.

2. **Center:** We initialize the resource allocations to an integer point around the center of the polytope $\{\sum_{i=1}^m K_i = K, K_i \geq 0\}$, which is $\left(\frac{K}{m}\right)\mathbf{1}$. This vector is rounded such that the players with lower indices round up, the players with higher indices round down and that the sum equals K.

3. **Avg:** We use the average of the output resource allocations of the first $m+1$ initializations (**ExtrPt** and **Center**) as the initial resource allocation. The average is rounded such that the players with lower indices round up, the players with higher indices round down and that the sum equals K.

4. **LP:** We run the CC-Lin algorithm using the LP relaxations as the concave approximation functions. We use the output of CC-Lin on LP relaxations as the initial resource allocation. Overall, this method requires CC-Lin to be run twice, once using the LP relaxations and once using the true objective functions.

For the following experiments, we restrict ourselves to knapsack problems described as

$$z_i(K_i) = \max\left\{c_i^T x_i : a_i^T x_i \leq K_i \ , \ x_i \in \{0,1\}^{N_i}\right\}$$

as arbitrary non-concave functions. We start with the following data set: We have $m = 10$ players and each player has $n = 10$ items. Item sizes are generated randomly such that $a_{ij} \sim U(1,5)$, and the item values depend on the item size, $c_{ij} \sim U(a_{ij}, 2a_{ij})$. The total amount of resource K is $(0.4)\sum_{i,j} a_{ij}$. We generate $T = 50$ such data sets. Table 2 shows the performance of each initialization methods on these data sets. For each data set t, Column

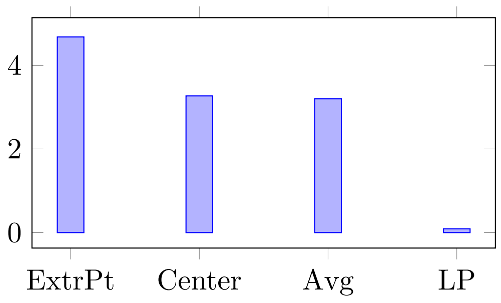

Figure 1: Average percent deviation of the initialization methods from the optimal solution for symmetric data.

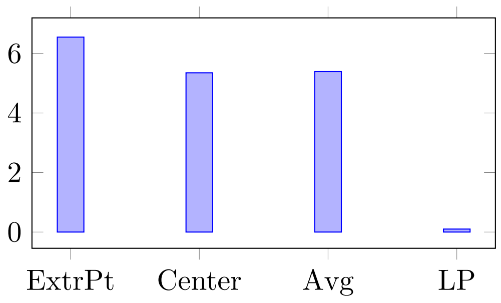

Figure 2: Average absolute percent deviation of the initialization methods from the optimal solution for asymmetric data.

2 shows the optimal solution. Columns 3-6 show the percent deviation of the output of initialization method A from the optimal solution, namely $\frac{\sum_{i=1}^{m} z_i^t(\bar{K}_i^{t,A}) - \sum_{i=1}^{m} z_i^t(K_i^{t*})}{\sum_{i=1}^{m} z_i^t(K_i^{t*})} \cdot 100$, where for initialization method A and data set t, $\bar{K}^{t,A}$ is the output of the CC-Lin algorithm using A, and K^{t*} is the optimal resource allocation. Figure 1 displays the absolute value of the average of these deviations taken over the T data sets for each method.

When the random items are distributed identically, the optimal resource allocation tends to be around the center of the feasible solution polytope. This means that the **Center** and **Avg** initialization methods terminate very quickly as their initial points are already close to the optimal solution. In order to get a better understanding of the performance of **Center** and **Avg**, we distort this structure in the following way: Every player is first assigned a number of valuable items, $\sim U(0, n)$. The item sizes are again $a_{ij} \sim U(1, 5)$, but now the value of a valuable item is $c_{ij} \sim U(5a_{ij}, 10a_{ij})$, and an low value item is $c_{ij} \sim U(1, 5a_{ij})$. The total amount of resource K is $(0.4)\sum_{i,j} a_{ij}$. We generate $T = 50$ such data sets. With this modification, we increase the deviation in the random optimal resource allocation. Table 3 shows the performance of each initialization methods on these data sets. For each data set t, Column 2 shows the optimal solution. Columns 3-6 show the percent deviation of the output of initialization method A from the optimal solution. Figure 2 displays the absolute value of the average of these deviations taken over the T data sets for each method.

It is clear that even with the increased deviation in the optimal resource allocations, the **LP** method for initial point selection dominates the others. After this observation, we test the robustness of the **LP** method. Namely, given a set of 100 items generated such that $a_{ij} \sim U(1, 5)$ and $c_{ij} \sim U(1, 10a_{ij})$, we distribute them among $m = 2, 5, 10, 20, 50$ players, and solve the resulting distributed problem using the **LP** initialization method. Table 4 shows the performance of each m value for each data set. For each data set t, Column 2

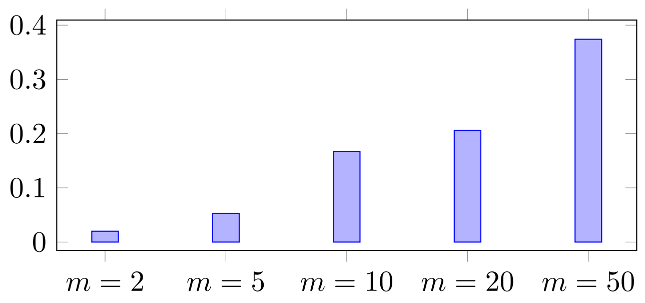

Figure 3: Average percent deviation of the **LP** initialization method from the optimal solution depending on different m values.

shows the optimal solution. Columns 3-7 show the percent deviation of the output of m value from the optimal solution. Figure 3 displays the absolute value of the average of these deviations taken over the T data sets for different number of players.

The results are not surprising. Let x^{LP} be the optimal solution of the linear relaxation, and x^* be the true optimal solution. Even though the vectors x^{LP} and x^* are not necessarily close to each other, as the items are distributed among more players, i.e. as m increases, for each player the difference between the optimal resource allocation $\sum_j a_{ij}x^*_{ij}$ and the linear relaxation resource allocation $\sum_j a_{ij}x^{LP}_{ij}$ increases. This difference causes the decrease in the performance of the algorithm. Overall, we can conclude that the **LP** initialization method outperforms the other three methods, due to its robustness. The average percent deviation from the optimal solution is less than 1 % for the **LP** initialization method.

5 Conclusions

We proposed decentralized algorithms for solving distributed integer programming problems, where every processor is assigned equal roles and there exists no central processor that has access to every player's data. Our algorithms are designed for settings with a cardinality constraint as the binding coupling constraint. The algorithms have optimality guarantees for some additional local constraint sets. We provide sufficient conditions for problems to guarantee optimality of the algorithms, and give examples of problems that satisfy these conditions. Furthermore, we provide error bounds for approximation results of our algorithms when applied to problems that don't satisfy the optimality condition. These approximations cover a wide range varying from heuristics and relaxations to noise and communication errors.

As future research, we focus on the same problem structure in an online setting. Namely, we assume that the coefficients change in every iteration and resource allocation is made prior to observing new data. We consider deterministic and randomized algorithms to solve this problem and provide their respective error bounds [12].

References

[1] E. H. Aghezzaf and L. A. Wolsey. Lot-sizing polyhedra with a cardinality constraint. *Operations Research Letters*, 11(1):13–18, 1992.

[2] D. P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1(1):7–66, 1992.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Athena Scientific, 1997.

[4] M. Bläser, T. Heynen, and B. Manthey. Adding cardinality constraints to integer programs with applications to maximum satisfiability. *Information Processing Letters*, 105(5):194–198, 2008.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[6] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. The power of randomization: Distributed submodular maximization on massive datasets. *CoRR*, abs/1502.02606, 2015.

[7] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, 2012.

[8] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions—ii. *Polyhedral combinatorics*, pages 73–87, 1978.

[9] A. Jadbabaie, A. Ozdaglar, and M. Zargham. A distributed newton method for network optimization. *48th IEEE Conference on Decision and Control*, pages 2736 – 2741, 2009.

[10] D. Jakovetic, J. Xavier, and J. M. F. Moura. Fast distributed gradient methods. *IEEE Transactions on Automatic Control*, 59(5):1131–1146, 2014.

[11] E. Karabulut. *Distributed Integer Programming.* PhD thesis, Georgia Institute of Technology, 2017.

[12] E. Karabulut, S. Ahmed, and G. Nemhauser. Decentralized online discrete optimization. *working paper.*

[13] B. Korte and D. Hausmann. An analysis of the greedy heuristic for independence systems. *Algorithmic aspects of combinatorics*, 2:65–74, 1978.

[14] J. Lee, M. Sviridenko, and J. Vondrák. Matroid matching: the power of local search. *SIAM Journal on Computing*, 42(1):357–379, 2013.

[15] I. Lobel, A. Ozdaglar, and D. Feijer. Distributed multi-agent optimization with state-dependent communication. *Mathematical Programming*, 129(2):255–284, 2011.

[16] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: identifying representative elements in massive data. *Advances in Neural Information Processing Systems 26*, 2013.

[17] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1):48–61, 2009.

[18] G. L. Nemhauser and L. A. Wolsey. *Wiley Series in Discrete Mathematics and Optimization : Integer and Combinatorial Optimization (1)*. Wiley-Interscience, Somerset, US, 1999.

[19] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[20] R. L. Raffard, C. J. Tomlin, and S. P. Boyd. Distributed optimization for cooperative agents: application to formation flight. *43rd IEEE Conference on Decision and Control*, 3:2453–2459, 2004.

[21] M. Rodeh. Finding the median distributively. *Journal of Computer and System Sciences*, 24:162–166, 1982.

[22] G. Singh and C. M. O'Keefe. Decentralised scheduling with confidentiality protection. *Operations Research Letters*, 44(4):514–519, 2016.

[23] M. Walter, P. Damcı-Kurt, S. S. Dey, and S. Küçükyavuz. On a cardinality-constrained transportation problem with market choice. *Operations Research Letters*, 44(2):170–173, 2016.

[24] E. Wei and A. Ozdaglar. Distributed alternating direction method of multipliers. *51st IEEE Conference on Decision and Control*, pages 5445 – 5450, 2012.

A Experimental Results

		Percent Deviation from Optimal			
Data Set	**Optimal**	**ExtrPt**	**Center**	**Avg**	**LP**
1	234	-6.41	-2.56	-4.27	-0.43
2	214	-4.21	-3.74	-4.21	0.00
3	236	-4.24	-3.81	-1.69	0.00
4	233	-3.00	-3.00	-2.15	0.00
5	230	-3.91	-3.04	-2.61	0.00
6	232	-4.74	-2.16	-5.17	0.00
7	234	-7.69	-2.99	-4.70	0.00
8	210	-2.86	-2.38	-2.86	0.00
9	220	-4.09	-1.36	-3.18	0.00
10	215	-4.19	-3.72	-4.65	0.00
11	231	-4.33	-2.16	-3.03	0.00
12	216	-6.02	-1.85	-4.17	0.00
13	250	-5.20	-4.80	-6.00	0.00
14	201	-2.99	-3.98	-4.48	0.00
15	234	-6.84	-5.13	-4.27	0.00
16	213	-2.35	-1.88	-1.88	0.00
17	221	-4.52	-2.71	-5.88	0.00
18	221	-4.98	-2.26	-4.07	0.00
19	225	-4.89	-3.11	-2.67	-0.44
20	232	-6.47	-3.02	-5.17	0.00
21	228	-5.26	-3.07	-2.63	-0.44
22	232	-6.03	-1.72	-2.59	-0.43
23	209	-4.31	-2.87	-2.87	0.00
24	240	-4.17	-3.33	-2.08	0.00
25	223	-4.93	-4.93	-2.69	0.00
26	224	-5.80	-4.46	-3.57	0.00
27	215	-4.19	-2.33	-4.19	-0.47
28	230	-9.13	-3.91	-6.52	0.00
29	224	-3.57	-3.57	-1.79	-0.45
30	225	-5.78	-5.78	-4.44	-0.44
31	217	-6.45	-5.99	-6.91	0.00
32	217	-2.76	-2.76	-1.84	0.00
33	210	-3.81	-2.86	-3.33	0.00
34	235	-5.11	-4.68	-2.55	0.00
35	228	-7.02	-5.70	-5.26	-0.44
36	217	-5.53	-3.23	-2.30	0.00
37	205	-4.39	-2.93	-1.95	0.00
38	205	-3.90	-1.95	-1.46	0.00
39	217	-4.61	-2.76	-2.76	-0.46
40	226	-4.42	-2.65	-1.33	0.00
41	214	-2.34	-4.67	-1.87	0.00
42	217	-5.53	-2.30	-4.15	-0.46
43	232	-4.31	-1.72	-1.29	0.00
44	225	-4.00	-3.56	-0.89	0.00
45	227	-3.96	-1.76	-3.52	0.00
46	229	-3.06	-5.68	-2.62	0.00
47	216	-3.70	-3.70	-1.85	0.00
48	217	-5.07	-4.15	-1.84	0.00
49	214	-3.74	-2.34	-0.93	0.00
50	219	-3.20	-2.28	-0.91	0.00

Table 2: Performance of the CC-Lin algorithm using different initialization methods for symmetric data.

		Percent Deviation from Optimal			
Data Set	**Optimal**	**ExtrPt**	**Center**	**Avg**	**LP**
1	1014	-7.40	-7.40	-5.72	-0.10
2	986	-7.20	-5.17	-3.45	0.00
3	918	-14.81	-1.31	-8.50	0.00
4	1068	-12.36	-11.52	-23.22	-0.19
5	934	-1.82	-7.71	-1.07	0.00
6	825	-1.70	-11.88	-3.03	0.00
7	1060	-8.30	-2.08	-5.28	-0.38
8	887	-0.68	-3.83	-0.45	-0.23
9	969	-16.72	-4.64	-10.32	0.00
10	1003	-0.90	-5.98	-1.30	0.00
11	978	-2.97	-2.66	-2.86	-0.20
12	1021	-9.11	-5.48	-5.88	-0.20
13	1028	-7.20	-3.99	-5.45	0.00
14	1171	-4.78	-1.79	-3.50	-0.09
15	1014	-4.24	-2.66	-6.80	0.00
16	1005	-6.17	-7.26	-5.67	0.00
17	945	-4.23	-3.39	-4.55	-0.11
18	1020	-8.14	-11.08	-8.63	-0.49
19	886	-6.55	-5.30	-6.32	-0.11
20	982	-2.55	-2.44	-0.81	-0.20
21	1065	-6.10	-4.69	-7.32	0.00
22	1092	-3.66	-3.30	-0.92	0.00
23	972	-6.89	-3.81	-8.44	0.00
24	957	-2.82	-9.30	-1.15	-0.31
25	960	-5.52	-7.92	-5.42	0.00
26	791	-5.06	-5.44	-3.16	0.00
27	994	-5.63	-8.45	-6.84	-0.30
28	933	-13.50	-6.00	-7.93	-0.11
29	1053	-3.04	-3.51	-1.52	0.00
30	890	-1.80	-4.38	-1.46	-0.11
31	1065	-8.08	-1.97	-5.73	-0.38
32	980	-5.10	-2.76	-0.71	-0.10
33	917	-5.67	-3.60	-5.89	-0.44
34	892	-6.39	-8.74	-5.38	-0.11
35	874	-12.47	-3.89	-5.84	0.00
36	983	-6.51	-4.88	-5.09	0.00
37	913	-2.74	-3.50	-3.50	0.00
38	768	-2.99	-4.56	-2.34	0.00
39	1019	-8.44	-5.40	-5.69	0.00
40	969	-4.64	-3.92	-2.79	-0.21
41	947	-5.60	-2.11	-2.75	0.00
42	964	-18.46	-5.50	-14.21	-0.21
43	988	-7.39	-7.79	-7.39	0.00
44	869	-10.24	-7.36	-6.44	0.00
45	1072	-5.32	-5.22	-8.12	0.00
46	942	-4.88	-8.81	-3.50	0.00
47	1023	-7.14	-2.25	-3.03	0.00
48	902	-8.98	-8.87	-6.76	-0.22
49	970	-5.98	-7.01	-8.97	0.00
50	901	-8.77	-4.88	-8.21	0.00

Table 3: Performance of the CC-Lin algorithm using different initialization methods for asymmetric data.

		Percent Deviation from Optimal				
Data Set	**Optimal**	$m = 2$	$m = 5$	$m = 10$	$m = 20$	$m = 50$
1	1030	0.00	-0.29	-0.78	-0.87	-0.87
2	1028	0.00	0.00	0.00	0.00	0.00
3	987	0.00	0.00	0.00	-0.20	-0.61
4	1038	0.00	-0.19	-0.58	-0.58	-0.58
5	1056	0.00	0.00	0.00	0.00	0.00
6	1020	0.00	0.00	-0.88	-1.57	-1.57
7	948	0.00	0.00	0.00	0.00	0.00
8	1008	0.00	0.00	0.00	0.00	0.00
9	1012	0.00	0.00	0.00	0.00	0.00
10	1043	0.00	0.00	0.00	0.00	0.00
11	981	0.00	0.00	0.00	0.00	-2.04
12	1007	0.00	0.00	0.00	0.00	-1.79
13	991	-0.10	0.00	0.00	0.00	0.00
14	1003	-0.20	-0.20	-0.50	-0.20	-0.20
15	881	0.00	0.00	0.00	0.00	0.00
16	1066	0.00	0.00	0.00	-0.47	0.00
17	1063	0.00	0.00	0.00	0.00	0.00
18	1035	0.00	0.00	0.00	0.00	-0.58
19	945	0.00	0.00	0.00	0.00	-0.85
20	986	-0.10	-0.20	-0.51	-0.61	0.00
21	955	-0.10	-0.31	-0.31	-0.10	-0.10
22	1008	0.00	-0.10	-0.30	0.00	0.00
23	968	0.00	0.00	0.00	0.00	0.00
24	920	0.00	0.00	-0.43	-0.54	0.00
25	1027	0.00	0.00	0.00	0.00	0.00
26	939	0.00	0.00	-0.43	-0.53	-0.53
27	927	0.00	-0.11	-0.11	-0.11	-1.40
28	1030	-0.10	-0.19	-0.19	-0.39	-0.10
29	944	0.00	0.00	0.00	0.00	0.00
30	876	0.00	0.00	0.00	0.00	0.00
Average	**990.73**	**-0.02**	**-0.05**	**-0.17**	**-0.21**	**-0.37**

Table 4: Performance of the CC-Lin algorithm using the **LP** initialization method for different m values.