A Faster Algorithm for the Resource Allocation Problem with Convex Cost Functions

Cong Shi*, Huanan Zhang*, Chao Qin†

* Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109

{shicong, zhanghn}@umich.edu

† Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208

chaoqin2019@u.northwestern.edu

Abstract

We revisit the classical resource allocation problem with general convex objective functions, subject to an integer knapsack constraint. This class of problems is fundamental in discrete optimization and arises in a wide variety of applications. In this paper, we propose a novel polynomial-time divide-and-conquer algorithm (called the multi-phase algorithm) and prove that it has a computational complexity of $\mathcal{O}(n \log n \log N)$, which outperforms the best known polynomial-time algorithm with $\mathcal{O}\big(n(\log N)^2\big)$.

Key words: polynomial-time algorithms; complexity analysis; data structure; nonlinear combinatorial optimization

History: Received January 2015; revision received June 2015; accepted July 2015.

1 Introduction

Consider the classical resource allocation problem which can be written as an integer programming problem as follows,

$$\min \sum_{i=1}^{n} f_i(x_i), \ \text{s.t.} \ \sum_{i=1}^{n} x_i = N, x_i \in \mathbb{Z}_0^+, \tag{1}$$

where $\mathbb{Z}_0^+$ denotes the set of nonnegative integers. We assume that each f_i is a convex function defined over $[0, N]$ and N is a positive integer. This problem is of particular interest when $N \gg n$, i.e., the number of resource units far exceeds the number of players. The compact physical description of the problem is to allocate a fixed homogeneous pool of N resource units in an optimal way to n distinct players so as to minimize the total allocation cost. This problem is commonly referred to as the *distribution of efforts problem.*

This problem is perhaps the simplest nonlinear combinatorial optimization problem, which has been extensively studied in Computer Science and Operations Research literature. Gross [10] first developed a simple greedy algorithm with computational complexity $\mathcal{O}(N \log n)$ to exactly solve this problem. The same type of greedy algorithm was then re-discovered by other researchers such as Fox [7] and Shih [20]. Subsequently, Katoh et al. [14] proposed an algorithm based on Lagrange multiplier methods, requiring $\mathcal{O}\big(n^2(\log N)^2\big)$ time. To this date, the best known polynomial-time algorithm for this problem runs in $\mathcal{O}\big(n(\log N)^2\big)$, which is due to the seminal work by Galil and Megiddo [8]. Many researchers have also focused on some variants of this basic problem (see, e.g., Dreyfus and Law [4], Weinstein and Yu [23], Dunstan [5]).

The key theoretical contribution of our work is to propose a novel polynomial-time algorithm (called the multi-phase algorithm) that runs in $\mathcal{O}(n \log n \log N)$, which represents a significant improvement in computational complexity over the existing literature. The key idea of the what-we-call multi-phase algorithm is to find the optimal partition of the marginal costs into several groups in each phase of our algorithm, in order to achieve the tightest complexity bound. It turns out that the optimal number of partitions in each phase is $\lceil en \rceil$ where $e \approx 2.71828$ is the base of the natural logarithm and the ceiling function $\lceil x \rceil$ is the smallest integer not less than x. Our algorithm is conceptually very simple and readily implementable in many practical settings.

This optimization problem finds various practical applications. In fact, our paper was primarily motivated by an important class of problems in inventory and supply chain management called the *joint replenishment problem* (JRP) (see, e.g., Khouja and Goyal [15] for an excellent review). More specifically, a warehouse manager wants to decide the total replenishment quantity and then allocate these ordered units to multiple (but non-identical) retailers. The problem studied in this paper serves as an important subroutine of the optimal allocation or distribution problem, given the total replenishment quantity in each replenishment cycle. Concisely speaking, the warehouse manager needs to determine how to allocate the fixed amount of goods to multiple retailers in order to minimize the sum of holding and backlogging costs (see the detailed model in Section 5).

Besides the aforementioned example, resource allocation problems are also abundant in many other application domains (see survey papers by Hochbaum [11] and by Patriksson [19]). Calinescu et al. [2] improved the quality of survey results with taking into account optimal resource planning. Federgruen and Groenevelt [6] applied greedy algorithms to network-based models. Veinott [21, 22] provided procedures of choosing the amounts of a single product to produce in each of a finite number of time periods in order to minimize the production and inventory carrying costs over the periods. Also, Johnson [12] and Karush [13] established methods to determine the optimal production program over time of a given commodity to minimize the total costs. Zipkin [24] applied different algorithms to find the optimal allocation for portfolio selection problems. Lee and Pierskalla [16] presented a mass screening program for computing the optimal test choice and screening periods among a fixed testing budget. The optimization model is also used in the optimum allocation problem in stratified sampling (see, e.g., Neyman [17]) and the optimal allocation problem for software-testing resources (see, e.g., Ohtera and Yamada [18]).

The remainder of this paper is organized as follows. Section 2 recalls the simple algorithm and shows its complexity bound. In Section 3, by proving a key lemma, we present and analyze a two-phase algorithm which lays the foundation for our multi-phase algorithm. In Section 4, we propose the multi-phase algorithm, and then prove its computational complexity bound. Section 5 is devoted to the numerical studies of our proposed algorithm. Finally, we conclude our paper and point out future research directions in Section 6.

Throughout the paper, we use the notation $\lfloor x \rfloor$ and $\lceil x \rceil$ frequently, where $\lfloor x \rfloor$ is defined as the largest integer value which is smaller than or equal to x; and $\lceil x \rceil$ is defined as the smallest integer value which is greater than or equal to x. Additionally, we denote $x^+ = \max\{x, 0\}$.

2 Naive Algorithm

To facilitate our discussion, we first present a *naive* algorithm that are conceptually very simple and natural. This motivates us to devise better strategies that ultimately lead to our proposed multi-phase algorithm.

For each $i = 1, \ldots, n, f_i$ is a convex function defined over the interval $[0, N]$. Based on incremental methods, for $x = 1, \ldots, N$, we define

$$g_i(x) \triangleq \Delta f_i(x) = f_i(x) - f_i(x-1)$$

to be the marginal cost for player i evaluated at integer point x_i. Due to convexity of our cost functions, we have $g_i(1) \leq g_i(2) \leq \ldots \leq g_i(N)$. Let G be the set of all marginal costs, i.e.,

$$G \triangleq \{g_i(x) : i = 1, \ldots, n, \text{ and } x = 1, \ldots, N\}.$$

Note that the cardinality $|G| = nN$. To facilitate the complexity analysis, we assume that each element in the set G has a distinct value, i.e., no two values in G are the same.

Remark We remark that if this is not the case, we can readily perturb the set G by adding some arbitrarily small positive numbers, e.g., we choose a sufficiently small positive δ and define the modified set $\tilde{G}$ of marginal costs by

$$\tilde{G} \triangleq \left\{ g_i(x) \triangleq \tilde{g}_i(x) + \delta^{(i-1)N+x} : i = 1, \ldots, n, \text{ and } x = 1, \ldots, N \right\}.$$

This perturbation makes sure that $g_i(x) \neq g_j(y)$ for all $i \neq j$ or $x \neq y$ and at the same time preserving the ordering in the original set G. This type of δ-perturbation method is also used to avoid degeneracy in solving a linear program (see Bertsimas and Tsitsiklis [1]).

Now we can reformulate the original resource allocation problem into finding the smallest N marginal costs in the set G. This is because we need to pick N resource units in optimization problem (1) with the goal to minimize the total costs, and picking each unit incurs a marginal cost that is included in the set G. Hence, an equivalent problem is to find the smallest N elements in the set G. Note that one may use the best sorting algorithm (e.g., quicksort) to sort G in an increasing order and simply pick the first N elements from the sorted set; however, this would require $\mathcal{O}(Nn \log(Nn))$, which is pseudo-polynomial. To this end, we need a much better algorithmic design.

We now present the *naive* algorithm in Algorithm 1. The naive algorithm employs a concept called *min-heap* (see, e.g., Cormen et al. [3]) to select the first N elements. The *(binary) heap* data structure is an array object that can be viewed as a nearly complete binary tree. Each node of the tree corresponds to an element of the array. The tree is completely filled on all levels except possibly the lowest, which is filled from the leftup to a point. In a min-heap, for every node i other than the root, the value of a node is at least the value of its parent. Thus, the smallest element in a min-heap is stored at the root, and the subtree rooted at a node contains values no smaller than that contained at the node itself.

In essence, what Algorithm 1 does is to push the next marginal cost of the minimum cost player (from the previous iteration) into the existing heap and select the one with the minimum cost in the current iteration. It turns out that Algorithm 1 has computational complexity of $\mathcal{O}(N \log n)$, which is formally stated in the following proposition.

Algorithm 1 Naive Algorithm

```
1: $x_i \leftarrow 1, i = 1, \ldots, n$;
2: Create a min-heap with $g_i(x_i), i = 1, \ldots, n$;
3: $r \leftarrow N$;
4: while $r > 0$ do
5:     Let $j = \underset{i}{\operatorname{argmin}}\, g_i(x_i)$ (i.e., the index of the root);
6:     Extract $g_j(x_j)$ from the heap (and maintain the min-heap property);
7:     $x_j \leftarrow x_j + 1$;
8:     Insert $g_j(x_j)$ into the heap (and maintain the min-heap property);
9:     $r \leftarrow r - 1$;
10: end while
```

Proposition 1. *For the classical resource allocation problems, the naive algorithm (i.e., Algorithm 1) has a computational complexity of* $\mathcal{O}(N \log n)$.

Proof. The heap initialization requires $\mathcal{O}(n)$, i.e., storing $g_i(1), i = 1, \ldots, n$ in a min-heap requires $\mathcal{O}(n)$. However, since $N \gg n$ in our problem, we drop the terms that do not involve an expression of N in the computational complexity bound.

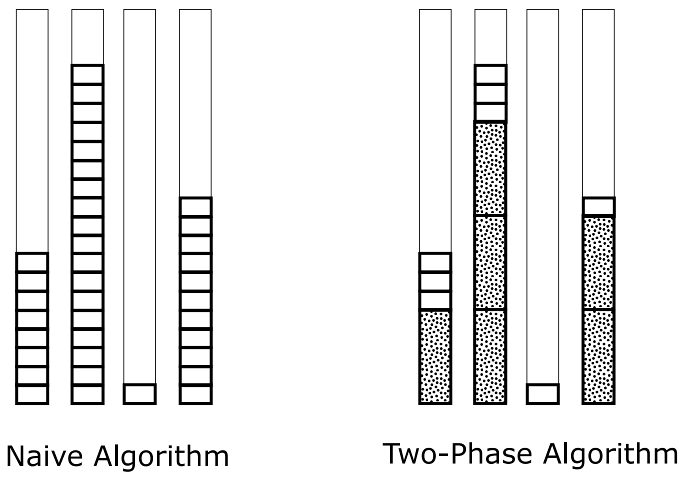
According to Algorithm 1, solving the problem requires N iterations in total. For each iteration, there is no need to build a fresh new min-heap, but deletion or addition for a min-heap is needed with complexity $\mathcal{O}(\log n)$. Thus, the complexity for the naive algorithm is $\mathcal{O}(N \log n)$. □

We notice that the major drawback of Algorithm 1 is that the computational complexity is exponential in the size of input, which is, strictly speaking, pseudo-polynomial (see, e.g., Garey and Johnson [9]). To be more specific, the complexity bound is exponential in $\log N$ which is the size of input. (Note that the integer N is stored in $\log N$ bits in computers.) To improve the computational efficiency, we first propose the following two-phase algorithm.

3 Two-Phase Algorithm

Before delving into details of the two-phase algorithm and proving that it gives the optimal solution, we first provide some high-level intuition. The key goal behind the algorithm is to reduce the number of evaluations needed to find an optimal solution. Because the marginal cost $g_i(x)$ is increasing in x (due to convexity properties of the objective functions), then if we know $g_i(x_i) \leq g_j(x_j)$, we can conclude that $g_i(x) \leq g_j(x_j)$ for all $x \leq x_i$. Thus, in the first phase, instead of evaluating every entry x_i, we may consider a block of entries and only evaluating the entry at the end of each block. Upon obtaining several blocks from the first phase, the remaining problem has a much smaller size and we can use the naive algorithm in the second phase. Figure 1 provides a graphical illustration of the two-phase algorithm. Each vertical bar represents a player i and each small element represents its marginal cost (with smaller ones at the bottom). The naive algorithm is inefficiently evaluating every single element while the two-phase algorithm attempts to first obtain the rough landscape of the optimal policy by evaluating only the last element of each (dotted) block (where the blocks have to be carefully chosen) and then gradually refine the solutions to the exact optimality.

Figure 1: Illustration of the two-phase algorithm



The detailed algorithm is explained as follows.

Phase 1. To employ a divide-and-conquer strategy, we first want to partition N marginal costs of each player into several groups. Then, instead of choosing the elements one by one in the naive algorithm, we obtain the elements group by group. As a result, we can confirm a large number of elements in the optimal solution in this phase. Then we obtain remaining ones in the left-over marginal costs by the naive algorithm.

We first divide N marginal costs of every player i ($i = 1, \dots, n$) into k (where $k \geq n$) groups with equal size. Each group has size of $m = \lfloor N/k \rfloor$ elements, which remains fixed throughout Phase 1. Next, for any $i = 1, \dots, n$ and $j = 1, \dots, k$, we define the set of marginal costs for each player i and each group j as follows,

$$G_{ij}(m) \triangleq \{g_i((j-1)m+1), g_i((j-1)m+2), \dots, g_i(jm)\}\,.$$

Note that the cardinality of $|G_{ij}(m)| = m$ for all i and j. Due to convexity of objective functions f_i, we know that for each player i in group j, the largest marginal cost is simply the end value of the group, i.e., $g_i(jm)$. For notational convenience, let set E be all of these end values, i.e.,

$$E(m) \triangleq \{g_i(jm) : i = 1, \dots, n, \text{ and } j = 1, \dots, k\}\,.$$

In Phase 1, we focus our attention solely on this set E. The intuition is that we can work on a much smaller set of marginal costs by reducing the original problem from size N to size k. It is easy to see that the cardinality of $|E(m)| = nk$. Now we run the naive algorithm on these nk values, and get the smallest $k - n + 1$ elements from the set $E(m)$ with complexity

$$\mathcal{O}((k-n+1)\log n) \leq \mathcal{O}(k \log n)\,.$$

Define the set of interest as

$$A(m) \triangleq \{\text{the smallest } k-n+1 \text{ elements in the set } E(m)\}\,.$$

We then proceed to construct the initial solution in Phase 1. The algorithm picks all the elements of the set $G_{ij}(m)$ whose end value $g_i(jm) \in A(m)$ thereby obtaining $m(k-n+1)$ elements. Next we shall show that these $m(k-n+1)$ elements are indeed in the optimal solution, which is the key lemma and building block of this paper.

Lemma 1. *When running the two-phase Algorithm, the $m(k-n+1)$ elements picked by the algorithm in Phase 1 are in the optimal solution for the resource allocation problem, where n is the number of players, k is the number of partitions of the N resource units, and m is the number of resource units in each partition.*

Proof of Lemma 1. We want to prove this statement by contradiction. We first focus on the set $A(m)$, i.e., the smallest $k-n+1$ elements taken from the set $E(m)$. Let index $l(i)$ denote the largest *group* index of each player i in the set $A(m)$. For example, if player 1 has 3 elements in the set $A(m)$, then $l(1) = 3$.

Without loss of generality, suppose the largest element in the set $A(m)$ (or the $(k-n+1)^{th}$ smallest element in the set $E(m)$) belongs to player 1's group. Then the element is simply $g_1(l(1)m)$. Let the largest value of marginal cost chosen in the optimal solution be c.

If $g_1(l(1)m)$ is not chosen in the optimal solution, that implies that $g_1(l(1)m) > c$. Since $g_1(l(1)m)$ is the $(k-n+1)^{th}$ smallest element in the set $E(m)$, and there are altogether $m(k-n+1)$ resource units from the original set G chosen based on the set $A(m)$, so at most $m(k-n+1)-1$ elements from the original set G whose values are smaller or equal to c.

Suppose that the optimal solution will pick all these $m(k-n+1)-1$ elements. We will show that the optimal solution does not have enough remaining elements to pick from the set G to obtain a feasible solution of size N. We claim that the optimal solution can pick at most additional $m-1$ elements from each player i $(i = 2, \ldots, n)$. Because $g_1(l(1)m) > c$, it is impossible to obtain any additional element from player 1. Without loss of generality, if the optimal solution picks m more elements from player 2, then $g_2(l(2)m+m) \in E(m)$ will be picked, and $g_2(l(2)m+m) > g_1(l(1)m)$, which contradicts the assumption that $g_1(l(1)m)$ is the largest elements in the set $A(m)$. Thus, at most additional $m-1$ elements from each player $i = 2, \ldots, n$ are in the optimal solution.

Therefore, based on $\lfloor x \rfloor \leq x$, the optimal solution can pick at most

$$m(k-n+1) - 1 + (m-1)(n-1) = mk - n < mk = \left\lfloor \frac{N}{k} \right\rfloor k \leq N,$$

which is not a feasible solution of size N. This completes the proof. □

Phase 2. In Phase 1, the algorithm has already picked $m(k-n+1)$ elements in the optimal solution, and remains to optimally pick additional $b \triangleq N - m(k-n+1)$ elements to complete the final optimal solution. Recall that the index $l(i)$ denote the largest *group* index of each player i in the set $A(m)$. For example, if player 1 has 3 elements in the set $A(m)$, then $l(1) = 3$. Next, we define the set $B_i(m,b)$ for each player i,

$$B_i(m,b) \triangleq \{g_i(l(i)m+1), g_i(l(i)m+2), \ldots, g_i(l(i)m+b)\},$$

where $|B_i(m,b)| = b = N - \left\lfloor \frac{N}{k} \right\rfloor (k-n+1)$. It is straightforward to check that the remaining b elements in the optimal solution is the smallest b elements in the set $B(m,b) \triangleq \bigcup_{i=1}^{n} B_i(m,b)$. If we run the naive algorithm for b iterations, we can get the smallest b elements in the set $B(m, b)$ with computational complexity

$$\begin{aligned}
\mathcal{O}(b \log n) &= \mathcal{O}\Bigg(\bigg(N - \left\lfloor \frac{N}{k} \right\rfloor (k - n + 1)\bigg) \log n\Bigg) \\
&\leq \mathcal{O}\Bigg(\bigg(N - \left(\frac{N}{k} - 1\right) (k - n + 1)\bigg) \log n\Bigg) \\
&\leq \mathcal{O}\Bigg(\left(k + \frac{Nn}{k}\right) \log n\Bigg) = \mathcal{O}\Bigg(\max\left(k, \frac{Nn}{k}\right) \log n\Bigg),
\end{aligned}$$

where the first inequality follows from the property of $\lfloor x \rfloor$ that $\lfloor x \rfloor > x - 1$; the second inequality follows from throwing away some insignificant terms; and the last equality follows from $k + \frac{Nn}{k} \leq 2 \max\left(k, \frac{Nn}{k}\right)$ where 2 is a constant and does not affect the order of the complexity.

Now, we have shown that the computational complexity bound for Phase 1 and Phase 2 are $\mathcal{O}(k \log n)$ and $\mathcal{O}\big(\max\left(k, \frac{Nn}{k}\right) \log n\big)$, respectively. Thus, the total complexity bound for the two-phase algorithm is

$$\begin{aligned}
&\mathcal{O}(k \log n) + \mathcal{O}\Bigg(\max\left(k, \frac{Nn}{k}\right) \log n\Bigg) \\
= \; &\mathcal{O}\Bigg(\max\left(k, \frac{Nn}{k}\right) \log n\Bigg),
\end{aligned}$$

where the equality follows from $\mathcal{O}(k \log n) \leq \mathcal{O}\big(\max(k, \frac{Nn}{k}) \log n\big)$.

Then we optimally choose the number of groups $k = \sqrt{Nn}$ that makes the two-phase algorithm runs in $\mathcal{O}\Big(\sqrt{Nn} \log n\Big)$. Indeed, it is reassuring to observe that both Phase 1 and Phase 2 share the same computational complexity if we choose to optimally divide each player's marginal costs into $k = \sqrt{Nn}$ groups.

Up to this point, we have derived the complexity bound for the two-phase algorithm, which is formally stated in the main theorem below.

Theorem 1. *For the classical resource allocation problems, the two-phase algorithm has a computational complexity of* $\mathcal{O}\Big(\sqrt{Nn} \log n\Big)$.

Theorem 1 forms a basis for our multi-phase algorithm described in Section 4. Before extending the current two-phase algorithm to a multi-phase algorithm, we first provide a simple numerical example in the following subsection.

3.1 A Simple Example

For clarity of presentation, we now manually run a simple example to better illustrate how our two-phase algorithm works. Suppose there are $n = 3$ players, and there are altogether $N = 12$

resource units. Moreover, suppose the cost functions are

$$\begin{aligned} f_1(x_1) &= \frac{x_1(x_1+1)}{2}, \\ f_2(x_2) &= x_2(x_2+1)+0.1, \\ f_3(x_3) &= \frac{3x_3(x_3+1)}{2}+0.2. \end{aligned}$$

Thus, $f_1(0)=0$, $f_2(0)=0.1$, $f_3(0)=0.2$, and the sets of marginal costs for these 3 players are $\{1,2,\dots,12\}$, $\{2,4,\dots,24\}$,and $\{3,6,\dots,36\}$, respectively. Thus,

$$G = \{\underbrace{1,2,\dots,12}_{G_1},\underbrace{2.1,4.1,\dots,24.1}_{G_2},\underbrace{3.2,6.2,\dots,36.2}_{G_3}\}.$$

Note that in this example, each element in G has a distinct value, and therefore no δ-perturbation procedure is required. Our objective is to pick the smallest 12 numbers in the set G.

Phase 1. In this phase, we choose $k=\sqrt{Nn}=6$, and then divide 12 elements of each players into 6 groups with equal size $m=\lfloor 12/6 \rfloor = 2$. More explicitly, the 6 groups are

$$\{1,2\},\{3,4\},\{5,6\},\{7,8\},\{9,10\},\{11,12\} \text{ for player 1;}$$

$$\{2.1,4.1\},\{6.1,8.1\},\{10.1,12.1\},\{14.1,16.1\},\{18.1,20.1\},\{22.1,24.1\} \text{ for player 2;}$$

$$\{3.2,6.2\},\{9.2,12.2\},\{15.2,18.2\},\{21.2,24.2\},\{27.2,30.2\},\{33.2,36.2\} \text{ for player 3.}$$

So our set E (taking the maximum value from each set above) is then

$$E = \{\underbrace{2,4,6,8,10,12}_{E_1},\underbrace{4.1,8.1,12.1,16.1,20.1,24.1}_{E_2},\underbrace{6.2,12.2,18.2,24.2,30.2,36.2}_{E_3}\}.$$

Phase 1 of our algorithm picks the smallest $k-n+1=4$ elements from the set E. We will need 4 iterations to extract these 4 elements. Initially, we push the first elements from E_1, E_2 and E_3 into a min-heap, i.e., push 2, 4.1 and 6.2 into a min-heap. We know that 2 being the smallest element is stored at the root of this min-heap.

(a) In Iteration 1, we pop the root of this min-heap 2, and then select the whole group associated with the end value 2, in this case, $\{1,2\}$ of player 1. Based on Lemma 1, they are in the optimal solution. Then we remove 2 from the min-heap, and insert the next element of E_1 (which is 4) into the heap. We notice that the modified heap has already maintained the min-heap property, and 4 from E_1 is at the root.

(b) In Iteration 2, we pop the root of the min-heap, which is 4 from E_1. Thus, the elements in the second group $\{3,4\}$ of player 1 should be in the optimal solution. Then we remove 4 from E_1 from the heap, and insert 6 from E_1 into the min-heap. The root will then be 4.1 from E_2.

(c) In Iteration 3, we pop the root of the min-heap, which is 4.1 from E_2, and the elements in the first group $\{2.1,4.1\}$ of player 2 are in the optimal solution. Then we remove 4.1 from E_2 from the heap, and insert 8.1 from E_2 into the min-heap. The root will then be 6 from E_1.

(d) In Iteration 4, we pop the root of the min-heap, which is 6 from E_1, and the elements in the third group $\{5,6\}$ of player 1 are in the optimal solution.

As a result, after completing Phase 1, we have picked $\{1, 2, 3, 4, 2.1, 4.1, 5, 6\}$, which must be in the optimal solution.

Phase 2. Since we have obtained 8 elements in the first phase, we need to get additional $12 - 8 = 4$ elements in the optimal solution. Now, the reduced sets of left-over resource elements for three players are $\{7, 8, \ldots, 12\}, \{6.1, 8.1, \ldots, 24.1\}$, and $\{3.2, 6.2, \ldots, 36.2\}$, respectively. Using the naive algorithm to obtain the remaining elements one by one, we can get $\{3.2, 6.1, 6.2, 7\}$ in the optimal solution.

Our two-phase algorithm yields $\{1, 2, 3, 4, 2.1, 4.1, 5, 6, 3.2, 6.1, 6.2, 7\}$ as the optimal solution, which are the smallest 12 marginal costs among three players. In addition, $f_1(0) = 0$, $f_2(0) = 0.1$, and $f_3(0) = 0.2$. The optimal objective value is the sum of the obtained marginal costs and $f_i(0)$ ($i = 1, 2, 3$), which is 50 in this case.

4 Multi-Phase Algorithm

Recall that in the two-phase algorithm, we first divide marginal costs of each player into several groups. In Phase 1, instead of optimally choosing elements one by one, we can choose the elements group by group. Then, in Phase 2, we obtain the remaining elements by working with a reduced size problem. In this section, we propose the multi-phase algorithm based on the exact same idea of the two-phase algorithm, and prove that it has a computational complexity of $\mathcal{O}(n \log n \log N)$.

Special Case: $N = n^s$. For simplicity, We first assume $N = n^s$ (where s is a positive integer) and then relax this assumption later. It is clear that $s = \frac{\log N}{\log n}$.

Suppose now we choose the number of groups $k = \lceil n^{1+\epsilon} \rceil$ where ϵ is some positive number (we will later show that the optimal $\epsilon^* = \frac{1}{\log n}$ in order to obtain the tightest possible bound). According to Phase 1 in the two-phase algorithm described in Section 3, we have already picked

$$\left\lfloor \frac{N}{k} \right\rfloor (k - n + 1)$$

number of elements in the optimal solution with computational complexity

$$\mathcal{O}((k - n + 1) \log n) \leq \mathcal{O}(k \log n) = \mathcal{O}\big(\lceil n^{1+\epsilon} \rceil \log n\big).$$

Thus, upon completion of the first phase, the number of elements remaining to pick is

$$\begin{aligned} N - \left\lfloor \frac{N}{k} \right\rfloor (k - n + 1) &< N - \left(\frac{N}{k} - 1\right)(k - n + 1) \\ &= \frac{Nn}{k} - \left(\frac{N}{k} - k\right) - (n - 1) \\ &< \frac{Nn}{k} = \frac{Nn}{\lceil n^{1+\epsilon} \rceil} \leq \frac{1}{n^\epsilon} N, \end{aligned}$$

where the second inequality is valid since $N \gg n$ and $k^* = \lceil en \rceil$ (which we will show later); and the first and the last inequalities hold using the properties of of $\lfloor x \rfloor$ and $\lceil x \rceil$.

Essentially, what Phase 1 does is to reduce the original problem from size N to a smaller size $\frac{1}{n^\epsilon}N$. Then in Phase 2, we use the same trick and divide each player into $k = \lceil n^{1+\epsilon} \rceil$ groups with the same computational complexity

$$\mathcal{O}\big(\lceil n^{1+\epsilon} \rceil \log n\big)\,.$$

Upon completion of Phase 2, the remaining elements to be picked is at most

$$\frac{1}{n^\epsilon}\left(\frac{1}{n^\epsilon}N\right) = \frac{1}{n^{2\epsilon}}N.$$

Thus, we can repeat the same procedure until the last phase. For each phase (with the exception of the last phase), the complexity stays unchanged, i.e., $\mathcal{O}\big(\lceil n^{1+\epsilon} \rceil \log n\big)$. Because it is assumed that $N = n^s$ in this special case, we repeat the above process for at most $\lfloor \frac{s}{\epsilon} \rfloor$ phases. After all these phases, we are unable to divide the reduced size into groups each with at least one element, and thus there are less than $\lceil n^{1+\epsilon} \rceil$ remaining elements to pick. By running the naive algorithm, we obtain these remaining elements with complexity $\mathcal{O}\big(\lceil n^{1+\epsilon} \rceil \log n\big)$.

Based on the complexity analysis above, the computational complexity for the multi-phase algorithm is therefore

$$\begin{aligned}
\mathcal{O}\Big(\Big\lfloor \frac{s}{\epsilon} \Big\rfloor \left(\lceil n^{1+\epsilon} \rceil \log n\right)\Big) &\leq \mathcal{O}\Big(\frac{s}{\epsilon} n^{1+\epsilon} \log n\Big) \\
&= \mathcal{O}\left(\frac{\log N}{\epsilon \log n} n^{1+\epsilon} \log n\right) \\
&= \mathcal{O}\left(\frac{n^{1+\epsilon} \log N}{\epsilon}\right).
\end{aligned}$$

To obtain the tightest computational complexity bound, we optimally choose $\epsilon^* = \frac{1}{\log n}$. By substituting $\epsilon^* = \frac{1}{\log n}$ into the complexity formula above for the multi-phase algorithm, we have

$$\mathcal{O}\left(\frac{n^{1+\epsilon^*} \log N}{\epsilon^*}\right) = \mathcal{O}(en \log n \log N) = \mathcal{O}(n \log n \log N)\,,$$

where the above equalities use $n^{\frac{1}{\log n}} = e$ (e is the base of the natural logarithm which does not change the order of the complexity). Up to this point, we have proven the special case where $N = n^s$(s is a positive integer).

General Case: General N. We shall then prove that the complexity bound $\mathcal{O}(n \log n \log N)$ remains true for general n and N. It follows from the following simple observation.

When $n^s < N < n^{s+1}$ (where s is a positive integer), because the upper bound for this particular N is n^{s+1}, the complexity is upper bounded by $\mathcal{O}\big(n \log n \log(n^{s+1})\big)$. In addition, we notice that $s < \frac{\log N}{\log n}$, which leads to $s + 1 < \frac{\log N}{\log n} + 1 < \frac{2\log N}{\log n}$. Therefore, the complexity can be written as follows,

$$\begin{aligned}
\mathcal{O}\big(n \log n \log\left(n^{s+1}\right)\big) &\leq \mathcal{O}\Big(n \log n \log\Big(n^{\frac{2\log N}{\log n}}\Big)\Big) \\
&= \mathcal{O}\left(n \log n \left(\frac{2\log N}{\log n}\right) \log n\right) \\
&= \mathcal{O}(n \log n \log N)\,,
\end{aligned}$$

where 2 is a constant which does not change the order of the complexity.

Thus, for general n and N, we have shown that the complexity bound for the multi-phase algorithm remains $\mathcal{O}(n \log n \log N)$, which is formally stated in the following theorem.

Theorem 2. *For the classical resource allocation problems, the multi-phase algorithm has a computational complexity of* $\mathcal{O}(n \log n \log N)$.

We observe that when $\epsilon^* = \frac{1}{\log n}$, the optimal number of groups $k^* = \lceil n^{1+\epsilon^*} \rceil = \lceil en \rceil$, which implies that it is optimal to divide each player's marginal costs into $\lceil en \rceil$ groups in each phase of the proposed multi-phase algorithm.

To summarize the multi-phase algorithm, we also present the pseudo-code for the multi-phase algorithm in Algorithm 2 below.

Algorithm 2 Multi-Phase Algorithm

```
1: $x_i \leftarrow 0, i = 1, \ldots, n$;
2: $r \leftarrow N, k \leftarrow \lceil en \rceil$, $m \leftarrow \lfloor \frac{r}{k} \rfloor$ ;
3: while $m \geq 1$ do
4:     $x_i \leftarrow x_i + m, i = 1, \ldots, n$;
5:     Create a min-heap with $g_i(x_i), i = 1, \ldots, n$;
6:     $t \leftarrow 1$;
7:     while $t \leq k - n + 1$ do
8:         Let $j = \operatorname*{argmin}_i g_i(x_i)$ (i.e., the index of the root);
9:         Extract $g_j(x_j)$ from the heap (and maintain the min-heap property);
10:        $x_j \leftarrow x_j + m$;
11:        Insert $g_j(x_j)$ into the heap (and maintain the min-heap property);
12:        $r \leftarrow r - m$;
13:        $t \leftarrow t + 1$;
14:    end while
15:    $x_i \leftarrow x_i - m, i = 1, \ldots, n$;
16:    $m \leftarrow \lfloor \frac{r}{k} \rfloor$;
17: end while
18: Run the naive algorithm to obtain the remaining $r$ elements in the optimal solution.
```

5 Computational Studies

To test the efficacy and efficiency of the proposed multi-phase algorithm, we carry out numerical experiments on an applied problem mentioned in Section 1, and report their running times. The performance metrics include the running time (T), the time reduction (T_reduct) compared with the best greedy algorithm in the literature, and the optimal objective value (OPT). All algorithms are implemented in Matlab R2014a on an Intel Core i7-3770 3.40GHz PC.

Joint replenishment problem

As mentioned in Section 1, the classical *joint replenishment problem* (JRP) arises in supply chain management where the manager has to decide the total replenishment quantity and then allocate these ordered units to multiple retailers with different costs. We refer interested readers to Khouja and Goyal [15] for a review of this class of problems. The problem studied in the paper serves as a subroutine of the optimal allocation or distribution problem, given the total replenishment quantity in each replenishment cycle. We can apply the multi-phase algorithm to find the optimal allocation for JRP.

We first briefly introduce the model. Let n be the number of retailers and N be the number of goods ordered at a particular replenishment point. Retailer i ($i = 1, \dots, n$) faces a stochastic demand D_i. Upon distribution of the N goods into n retailers, the demand realizes and the system incurs holding and backlogging costs. Let h_i and b_i be the per-unit holding cost of excess inventory and per-unit backlogging cost of unsatisfied demand units for retailer i. The decision variable q_i is defined as the number of goods shipped from warehouse to retailer i.

We assume that demand D_i follows a truncated normal distribution defined on interval $[0, \infty)$ with mean $\mu_i > 0$ and standard deviation σ_i. We write its probability density function as $f(x; \mu_i, \sigma_i, 0, \infty)$. Then the expected cost $c_i(q_i)$ for retailer i is

$$c_i(q_i) = \mathbb{E}\left[h_i(q_i - D_i)^+ + b_i(D_i - q_i)^+\right].$$

It is well-known that this *newsvendor* cost $c_i(q_i)$ is convex in terms of q_i. Our objective is to minimize the total expected system-wide cost, i.e.,

$$\min \sum_{i=1}^{n} c_i(q_i), \ \text{ s.t. } \sum_{i=1}^{n} q_i = N, \ q_i \in \mathbb{Z}_0^+.$$

We apply the multi-phase algorithm to solve this problem. Here are our choices of parameters. We set $n \in \{3, 6, 9\}$ and $N \in \{100, 1000, 10000, 100000\}$, and $b_i = 2h_i = 2i$ ($i = 1, \dots, n$). For each fixed pair (n, N), we carry out six experiments with different choices of demand parameters μ_i and σ_i. The results are summarized in Table 1.

As shown in Table 1, we report the results of the aforementioned six experiments for each (n, N) pair. The average running time of solving this problem is extremely fast, less than 0.1 seconds even when N reaches 10^5. In addition, as N increases to 10^5, the average time reduction of the multi-phase algorithm increases up to 99.84%.

6 Conclusion

In this paper, we revisited the classical resource allocation problem with general convex functions, which is fundamental in optimization theory and widely applicable in practice. We proposed a novel algorithm called *multi-phase* algorithm and analyzed its computational complexity. We demonstrated, once again, the power of divide-and-conquer ideas combined with exploiting the special problem structure. One promising future research direction is to consider non-separable convex objective functions in which new ideas and techniques are needed to be developed.

Table 1: Performance of the multi-phase algorithm for the joint replenishment problem

Running Time (T), Time Reduction (T_reduct), Optimal Objective Value (OPT)

n		3			6			9		
N		T	T_reduct	OPT	T	T_reduct	OPT	T	T_reduct	OPT
10^2	1	0.007	82.00%	24	0.017	61.36%	41	0.025	54.55%	62
	2	0.006	76.00%	28	0.017	57.50%	52	0.025	53.70%	76
	3	0.006	76.92%	20	0.016	60.98%	32	0.026	52.73%	47
	4	0.006	75.00%	285	0.017	58.54%	448	0.030	50.00%	614
	5	0.007	72.00%	360	0.018	56.10%	656	0.029	49.12%	953
	6	0.006	76.00%	226	0.017	57.50%	327	0.028	50.88%	425
	avg	**0.006**	**76.32%**	**157**	**0.017**	**58.66%**	**259**	**0.027**	**51.83%**	**363**
10^3	1	0.056	90.28%	444	0.053	90.42%	475	0.049	91.06%	579
	2	0.059	89.97%	503	0.054	90.63%	588	0.047	91.30%	742
	3	0.055	90.32%	388	0.053	90.55%	369	0.051	90.81%	425
	4	0.057	90.31%	3284	0.057	89.89%	4668	0.055	90.23%	6139
	5	0.057	90.09%	4042	0.057	90.02%	6793	0.058	89.79%	9534
	6	0.056	90.21%	2557	0.060	90.00%	3396	0.057	89.86%	4232
	avg	**0.057**	**90.19%**	**1870**	**0.056**	**90.25%**	**2715**	**0.053**	**90.51%**	**3609**
10^4	1	0.072	98.74%	4410	0.067	98.80%	4724	0.064	98.84%	5789
	2	0.070	98.77%	4998	0.066	98.81%	5853	0.064	98.82%	7413
	3	0.070	98.77%	3849	0.071	98.74%	3669	0.064	98.85%	4248
	4	0.071	98.77%	32803	0.073	98.72%	46661	0.069	98.79%	61394
	5	0.072	98.76%	40391	0.073	98.73%	67905	0.074	98.72%	95343
	6	0.070	98.78%	25536	0.073	98.74%	33932	0.072	98.74%	42321
	avg	**0.071**	**98.76%**	**18664**	**0.070**	**98.76%**	**27124**	**0.068**	**98.79%**	**36085**
10^5	1	0.092	99.84%	44094	0.092	99.84%	47239	0.078	99.86%	57894
	2	0.093	99.84%	49980	0.088	99.84%	58522	0.084	99.84%	204708
	3	0.093	99.84%	38487	0.086	99.85%	36689	0.082	99.85%	131305
	4	0.095	99.84%	328029	0.095	99.83%	466613	0.088	99.84%	613936
	5	0.096	99.83%	413997	0.098	99.83%	852023	0.098	99.83%	953435
	6	0.093	99.84%	255354	0.095	99.83%	339313	0.093	99.84%	504853
	avg	**0.094**	**99.84%**	**188323**	**0.092**	**99.84%**	**300066**	**0.087**	**99.84%**	**411022**

Acknowledgment

The authors thank the Area Editor Professor Iain A. Stewart, and three anonymous referees for their constructive comments and suggestions, which helped improve both the content and the exposition of this paper. The work is partially supported by NSF grants CMMI-1362619 and CMMI-1451078.

References

[1] Bertsimas, D., J. N. Tsitsiklis. 1997. *Introduction to linear optimization*, vol. 6. Athena Scientific Belmont, MA.

[2] Calinescu, M., S. Bhulai, B. Schouten. 2013. Optimal resource allocation in survey designs. *European Journal of Operational Research* **226**(1) 115 – 121.

[3] Cormen, T. H., C. E. Leiserson, R. L. Rivest, C. Stein. 2009. *Introduction to Algorithms*. The MIT Press, Cambridge, MA.

[4] Dreyfus, S. E., A. M. Law. 1977. *The Art of Theory of Dynamic Programming*. Academic Press, Orlando, FL.

[5] Dunstan, F. D. J. 1977. An algorithm for solving a resource allocation problem. *Operational Research Quarterly* **28**(4) 839–851.

[6] Federgruen, A., H. Groenevelt. 1986. The greedy procedure for resource allocation problems: Necessary and sufficient conditions for optimality. *Operations Research* **34**(6) 909–918.

[7] Fox, B. 1966. Discrete optimization via marginal analysis. *Management Science* **13**(3) pp. 210–216.

[8] Galil, Z., N. Megiddo. 1979. A fast selection algorithm and the problem of optimum distribution of effort. *J. ACM* **26**(1) 58–64.

[9] Garey, M. R., D. S. Johnson. 1978. "Strong" NP-completeness results: motivation, examples and implications. *J. ACM* **25** 499–508.

[10] Gross, O. 1956. A class of discrete type minimization problems. Research Memorandum No. 1644, Santa Monica California: Rand Corporation.

[11] Hochbaum, D.S. 2007. Complexity and algorithms for nonlinear optimization problems. *Annals of Operations Research* **153**(1) 257–296.

[12] Johnson, S. M. 1957. Sequential production planning over time at minimum cost. *Management Science* **3**(4) 435–437.

[13] Karush, W. 1958. On a class of minimum-cost problems. *Management Science* **4**(2) 136–153.

[14] Katoh, N., T. Ibaraki, H. Mine. 1979. A polynomial time algorithm for the resource allocation problem with a convex objective function. *The Journal of Operations Research Society* **30**(5) 449–455.

[15] Khouja, M., S. Goyal. 2008. A review of joint replenishment problem literature: 1989-2005. *European Journal of Operational Research* **186** 1–16.

[16] Lee, H. L., W. P. Pierskalla. 1988. Mass screening models for contagious diseases with no latent period. *Operations Research* **36**(6) pp. 917–928.

[17] Neyman, J. 1934. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society* **97** 558–606.

[18] Ohtera, H., S. Yamada. 1990. Optimal allocation and control problems for software-testing resources. *IEEE Transactions on Reliability* **39**(2) 171–176.

[19] Patriksson, M. 2008. A survey on the continuous nonlinear resource allocation problem. *European Journal of Operational Research* **185**(1) 1–46.

[20] Shih, W. 1974. A new application of incremental analysis in resource allocations. *Operational Research Quarterly* **25**(4) pp. 587–597.

[21] Veinott, F. 1964. Production planning with convex costs: A parametric study. *Management Science* **10**(3) 441–460.

[22] Veinott, F. 1966. The status of mathematical inventory theory. *Management Science* **12**(11) 745–777.

[23] Weinstein, I. J., O. S. Yu. 1973. Comment on an integer maximization problem. *Operations Research* **21**(2) 648–650.

[24] Zipkin, P. H. 1980. Simple ranking methods for allocation of one resource. *Management Science* **26**(1) pp. 34–43.