Journal of Discrete Algorithms 34 (2015) 137–146

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A faster algorithm for the resource allocation problem with convex cost functions

Cong Shi [a,*], Huanan Zhang [a], Chao Qin [b]

[a] Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109, United States
[b] Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, United States

ARTICLE INFO

Article history:
Received 26 January 2015
Received in revised form 24 June 2015
Accepted 24 June 2015
Available online 2 July 2015

Keywords:
Polynomial-time algorithms
Complexity analysis
Data structure
Nonlinear combinatorial optimization

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 $\mathcal{O}\left(n(\log N)^2\right)$.

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 $\mathcal{O}\left(n^2(\log N)^2\right)$ time. To this date, the best known polynomial-time algorithm for this problem runs in $\mathcal{O}\left(n(\log N)^2\right)$, 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]).

* Corresponding author.
E-mail addresses: shicong@umich.edu (C. Shi), zhanghn@umich.edu (H. Zhang), chaoqin2019@u.northwestern.edu (C. Qin).

http://dx.doi.org/10.1016/j.jda.2015.06.001
1570-8667/

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 is 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]).