A new combinatorial algorithm for separable convex resource allocation with nested bound constraints

Zeyang Wu
Department of Industrial and Systems Engineering, University of Minnesota, wuxx1164@umn.edu.

Kameng Nip
School of Mathematical Sciences, Xiamen University, China, kmnip2004@gmail.com.

Qie He
Department of Industrial and Systems Engineering, University of Minnesota, qiehe01@gmail.com. Corresponding author.

The separable convex resource allocation problem with nested bound constraints aims to allocate B units of resources to n activities to minimize a separable convex cost function, with lower and upper bounds on the total amount of resources that can be consumed by nested subsets of activities. We develop a new combinatorial algorithm to solve this model exactly. Our algorithm is capable of solving instances with millions of activities in several minutes. The running time of our algorithm is at most 73% of the running time of the current best algorithm for benchmark instances with three classes of convex objectives. The efficiency of our algorithm derives from a combination of constraint relaxation and divide and conquer based on infeasibility information. In particular, nested bound constraints are relaxed first; if the obtained solution violates some bound constraints, we show that the problem can be divided into two subproblems of the same structure and smaller sizes, according to the bound constraint with the largest violation.

Key words: resource allocation; constraint relaxation; divide and conquer; polynomial-time algorithms; mixed-integer convex optimization

1. Introduction

The resource allocation problem is a collection of optimization models with a wide range of applications in production planning, logistics, portfolio management, telecommunication, statistical surveys, and machine learning (Ibaraki and Katoh 1988, Katoh et al. 2013, Patriksson 2008, Patriksson and Strömberg 2015). In its simplest case, the goal is to allocate certain units of resources to a set of activities to minimize the total allocation cost. In this paper, we study the model with a separable convex cost function and prescribed lower and upper bounds on the total amount of resources that could be consumed by some nested subsets of activities. These bounds could represent storage limits, time-window requirements, or budget constraints, depending on the application. We called this model the *resource allocation problem with nested lower and upper bound constraints*. This model also appears as a subproblem in vehicle routing models in green logistics (Kramer et al. 2015, Fukasawa et al. 2018) and the support vector ordinal regression problem (Chu and Keerthi 2007, Vidal et al. 2018), and has to be solved repeatedly by iterative algorithms for these models. All these applications motivate us to develop an efficient algorithm for this model.

We now introduce the mathematical programming formulation of this model. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$ for any positive integer n and $\emptyset$ otherwise. The resource allocation problem with nested lower and upper bound constraints can be formulated as the following mixed-integer convex program:

$$
\begin{aligned}
\min_{x} \quad & f(x) := \sum_{i=1}^{n} f_i(x_i) && && \text{(1a)}\\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i = B, && && \text{(1b)}\\
& a_i \le \sum_{k=1}^{i} x_k \le b_i, && \forall i \in [n-1], && \text{(1c)}\\
& 0 \le x_i \le d_i, && \forall i \in [n], && \text{(1d)}\\
& x_i \in \mathbb{Z}, && \forall i \in [n], && \text{(1e)}
\end{aligned}
$$

where n is the number of activities in consideration, f is a separable cost function with f_i being the univariate convex cost function for activity $i \in [n]$, B is the total units of resources to allocate, $a_i, b_i \ge 0$ for $i \in [n-1]$, $d_i > 0$ for $i \in [n]$, and $\mathbb{Z}$ is the set of integers. Formulation (1) stipulates that we want to determine the amount of resources x_i allocated to activity i to minimize the total allocation cost, while satisfying the constraint (1b) on the total amount resources consumed, the bound constraints (1c) on amount of resources that can be consumed by the first i activities, simple bound constraints (1d), and integrality constraints (1e). We use DRAP-NC to denote the discrete model described by Formulation (1), and RAP-NC to denote its continuous relaxation, i.e., Formulation (1) without constraints (1e).

Vidal et al. (2018) first studied DRAP-NC and RAP-NC, and called constraints (1c) the nested lower and upper bound constraints, since these constraints are imposed upon a sequence of nested subsets of activities. As illustrated in Vidal et al. (2018), these nested bound constraints are motivated by a variety of applications including production planning, vessel fuel optimization, and stratified sampling. RAP-NC also appears as a subproblem in iterative algorithms for other more complex optimization and machine learning models, such as the pollution-routing model (Bektaş and Laporte 2011) and the joint vehicle routing and speed optimization model (Fukasawa et al. 2018) in green logistics and the support vector ordinal regression problem (Chu and Keerthi 2007, Vidal et al. 2018) in supervised learning. Vidal et al. (2018) developed an efficient exact algorithm called the Monotonic Decomposition Algorithm (MDA) to solve DRAP-NC. Its running time is $O(n \log n \log B)$, currently the best in the literature. MDA could also be used to find an ϵ-optimal solution of RAP-NC (a solution x satisfying $\|x - x^*\|_\infty \leq \epsilon$ with x^* being some optimal solution) in $O(n \log n \log \frac{nB}{\epsilon})$ time through parameter scaling.

In this paper, we introduce an infeasibility guided divide-and-conquer algorithm to solve DRAP-NC exactly. We call this algorithm DCA for short in the rest of this paper. The worst-case running time of DCA is $\Theta(\max\{n^2 \log \frac{B}{n}, n^2\})$. Although this running time is much worse than that of MDA, the computational performance of DCA is surprisingly good on a variety of instances from different applications. *In particular, the running time of DCA is between* 33% *and* 73% *of the running time of MDA on all the test instances (with the largest instance having more than 6.5 million variables). The running time of DCA also grows much slower with* n^2 *on all the test instances.* We hypothesize that this mismatch between the theoretical running time and practical performance of DCA is largely due to the conservative worst-case analysis of the complexity of the algorithm. In particular, the asymptotic complexity result indicates that there exist "bad" instances for which the running time of DCA is proportional to the maximum of $n^2 \log \frac{B}{n}$ and n^2 for large n or B. But it seems that all the benchmark instances in the literature are not bad enough for the running time to reach the asymptotic bound. In Section 6, we report one statistic, the number of resource allocation subproblems solved by DCA, in all computational experiments to provide some evidence for our hypothesis. In addition, we introduce in Section 5 a family of DRAP-NC instances for which DCA only needs two recursions to terminate, illustrating the advantage of DCA in solving some instances. Finally, our algorithm can be easily extended to find an ϵ-approximate solution of RAP-NC by scaling all parameters by $\frac{n}{\epsilon}$, due to a proximity result from Hochbaum (1994).

We summarize the contributions of this paper below.

1. We develop a new combinatorial algorithm DCA to solve the discrete resource allocation problem with nested lower and upper bound constraints. Its worst-case running time is $\Theta(\max\{n^2 \log \frac{B}{n}, n^2\})$. DCA can also be extended to find an ϵ-optimal solution of RAP-NC with a worst-case running time $\Theta(\max\{n^2 \log \frac{B}{\epsilon}, n^2\})$.

2. We apply DCA to a variety of benchmark instances of DRAP-NC with different convex objectives. DCA is efficient in solving all these instances, with the running time being between 33% and 73% of the running time of the currently best algorithm MDA.
3. We present some explanation on the practical efficiency of DCA, and introduce specially designed instances for which the running time of DCA varies significantly.

The remainder of this section is organized as follows. Section 2 reviews existing algorithms for DRAP-NC and similar models that are mostly related to our algorithm. Section 3 introduces the details of DCA and Section 4 presents the proof of its correctness. In Section 5, we analyze the time complexity of DCA, and present some specially designed instances for which DCA only requires a few recursions to terminate. We present extensive numerical results on DRAP-NC instances in Section 6 and conclude in Section 7.

2. Literature review

Resource allocation is a fundamental problem in operations research with a wide variety of applications, from early examples in distribution of search effort (Koopman 1953) and sample allocation in stratified sampling (Neyman 1934) to recent ones including vessel speed optimization (He et al. 2017), energy management (van der Klauw et al. 2017), and machine learning (Vidal et al. 2018). Depending on the context of the applications, the resource could be divisible or indivisible, leading to continuous or discrete resource allocation models. The methodologies for these two classes of models could be significantly different. In theory, algorithms for many discrete models could be easily applied to find an ϵ-optimal solution of the continuous models by scaling the parameters. On the other hand, a large class of algorithms for the continuous models rely on optimality conditions such as the Karush-Kuhn-Tucker conditions, which do not have a counterpart in the discrete cases. In this paper, we focus on surveying results for discrete resource allocation models. We only mention one paper on the continuous model that is closely connected to our algorithm: van der Klauw et al. (2017) used the same divide-and-conquer framework to solve RAP-NC, which is a convex optimization problem. Since they were handling continuous problems, the proof of correctness of their algorithm used Karush-Kuhn-Tucker conditions of the original and relaxed problems. For a comprehensive review of algorithms for continuous resource allocation models, we refer interested readers to the following excellent surveys (Patriksson 2008, Katoh et al. 2013, Patriksson and Strömberg 2015).

Discrete resource allocation problems are NP-hard in general, but efficient algorithms exist when the objective function and the resource constraints have certain properties. One such case is that the objective function is separable and convex and there are no additional resource constraints other than the non-negativity constraints, i.e., model (1) without constraints (1c) and upper bounds on variables in constraints (1d). This model (or its continuous relaxation) is usually called the simple resource allocation problem in the literature. We use DRAP to denote this model. Gross (1956) developed the first greedy algorithm for DRAP, with a running time of $O(B\log n + n)$. With more complex greedy steps, polynomial-time algorithms have been developed (Galil and Megiddo 1979, Frederickson and Johnson 1982). The current best algorithm was developed by Frederickson and Johnson (1982) and runs in $\Theta(\max\{n\log\frac{B}{n}, n\})$ time. Hochbaum (1994) developed a scaling-based algorithm with a greedy subroutine for a more general resource allocation problem. Her algorithm, when specialized to DRAP, also achieves the best running time $\Theta(\max\{n\log\frac{B}{n}, n\})$, but avoids many complicated procedures of the algorithm by Frederickson and Johnson (1982). This is also the algorithm we use in DCA to solve the subproblem DRAP. Faster algorithms exist for DRAP with special objective functions: DRAP with linear or quadratic separable objective functions can be solved in $O(n)$ time (Brucker 1984). It should be noted that all the aforementioned algorithms for DRAP can be easily extended to DRAP with additional upper bounds on variables (Ibaraki and Katoh 1988).

A model that is slightly more complicated is DRAP with additional nested upper bound constraints, i.e., model (1) with $a_i = -\infty$ in constraints (1c) for each $i \in [n]$. This type of nested upper bound constraints are also called ascending constraints or cumulative constraints in the literature. Dyer et al. (1984) proposed an early polynomial-time divide-and-conquer algorithm with a running time of $O(n\log n\log^2\frac{B}{n})$. The scaling-based algorithm by Hochbaum (1994) could also be applied to solve this model and runs in $O(n\log n\log\frac{B}{n})$ time. Vidal et al. (2016) developed an $O(n\log m\log\frac{B}{n})$-time divide-and-conquer algorithm, provided that there are m (out of $n-1$) nested upper bound constraints. Their algorithm recursively divides a problem into two subproblems of even size, and uses optimal solutions of the subproblems to tighten variable bounds so that each subproblem only needs to be solved as a DRAP. The same framework was later extended in Vidal et al. (2018) to deal with DRAP-NC.

The nested upper bound constraints are a special case of the more general submodular constraints. The submodular constraint has the form of $\sum_{i\in S} x_i \leq r(S)$, where the set $S \subseteq [n]$ is an element of a distributive lattice and r is a submodular function defined over this lattice. The submodular constraints generalize a large class of constraints such as the generalized upper bound constraints, tree constraints, and network constraints (Katoh et al. 2013). Federgruen and Groenevelt (1986a) showed that the greedy approach for DRAP in Gross (1956) could be extended to DRAP with submodular constraints. The running time of this greedy approach is $O(B(\log n + F))$ where F is the number of operations required to check the feasibility of a given increment in the solution. Groenevelt (1991) developed two polynomial-time algorithms for DRAP with certain types of submodular constraints: a decomposition algorithm which runs in polynomial time for network and generalized symmetric polymatroids, and a bottom up algorithm which runs in polynomial time when the polymatroid is given as an explicit list of constraints. The decomposition algorithm uses optimal solutions of DRAP to tighten the variable bounds of the original problem. Similar ideas have been used in the decomposition algorithms for DRAP with nested upper bound constraints (Vidal et al. 2016) and DRAP-NC (Vidal et al. 2018). Hochbaum (1994) developed an $O(n(\log n + F)\log \frac{B}{n})$-time scaling-based algorithm that uses a greedy algorithm as a subroutine for DRAP with submodular constraints. This improvement on the time complexity is based on a general proximity theorem between DRAP with submodular constraints and its scaled version. The proximity theorem showed that the greedy algorithm can be applied to the model with arbitrary increments and only the last increment of each variable can be potentially erroneous, but the last increment could be removed to produce a valid lower bound of the integer optimal solution. Hochbaum (1994) also showed that the algorithm runs in $O(n(\log n + F)\log \frac{B}{\epsilon})$ time for the continuous relaxation of DRAP with submodular constraints, and cannot be improved for DRAP and DRAP with generalized upper bounds.

There has been only limited work focusing on problems with resource constraints with nested lower and upper bound constraints. Ahuja and Hochbaum (2008) studied a dynamic lot-sizing model with linear costs and nested lower and upper bound constraints on the cumulative production. They treated the problem as a min-cost flow problem over a network with a special structure, and developed an $O(n \log n)$-time algorithm using efficient data structures for the successive shortest path algorithm. *We would like to point out that* *the nested lower and upper bound constraints are indeed a special case of the submodular constraints. This connection is not obvious, and to the best of our knowledge, has not been mentioned in the literature.* The reason is that one could show that the nested lower and upper bound constraints are a special case of the network constraints (see the detailed definition of network constraints in Section 2.2 of Katoh et al. (2013)), and network constraints have been shown to be a special case of the submodular constraints (Federgruen and Groenevelt 1986b). Therefore, Hochbaum's algorithm for DRAP with submodular constraints could be applied to DRAP-NC. After working on the details of specializing Hochbaum's algorithm to DRAP-NC, one could show that the algorithm actually runs in $O(n^2 \log \frac{B}{n})$ time when $B > n$. In their recent paper, Vidal et al. (2018) showed that their decomposition algorithm for DRAP with nested upper bound constraints can be extended to DRAP-NC with nontrivial bound tightening techniques. Their algorithm MDA runs in $O(n \log m \log B)$ time, where m is the pairs of nested lower and upper bound constraints in (1c), and can be applied to the continuous relaxation in $O(n \log m \log \frac{nB}{\epsilon})$ time. Their algorithm currently has the best time complexity for DRAP-NC. Although MDA also uses a divide-and-conquer framework, unlike DCA, it always divides the problem evenly at each recursion.

Finally, DRAP-NC belongs to the class of convex mixed-integer nonlinear programs (MINLPs), so it could be solved by a general MINLP solver. Please refer to Kronqvist et al. (2019) for the latest survey of methods and solvers for convex MINLPs. Furthermore, DRAP-NC has a separable convex objective function. Several reformulation and approximation techniques that explore this structure (Hijazi et al. 2014, Kronqvist et al. 2018) could be applied to improve the performance when a general MINLP solver is used to solve DRAP-NC.

3. An infeasibility guided divide-and-conquer algorithm

Before introducing our algorithm for DRAP-NC, we mention several assumptions used throughout the paper.

1. Without loss of generality, we assume that all data involved in (1b)-(1d) are integral, $a_1 \leq a_2 \leq \ldots \leq a_{n-1}$, $b_1 \leq b_2 \leq \ldots \leq b_{n-1}$, and the lower bound for each variable is 0. We also assume that $a_i < b_i$ for $i \in [n-1]$. For if $a_i = b_i$ for some i, we can break the original problem into one DRAP-NC with variables $x_1, \ldots, x_i$ and one DRAP-NC with variables $x_{i+1}, \ldots, x_n$ and solve them separately.

2. We assume that f_i is convex over the interval $[0, d_i]$ and is encoded by a value oracle (which returns the value of $f_i(x)$ for any given input x in one operation) for each $i \in [n]$. We do not impose any additional assumption on f_i such as differentiability or Lipschitz continuity.
3. We assume the problem is feasible. Feasibility can be checked easily by checking whether $[a_i, b_i] \cap [0, \sum_{k=1}^{i} d_k] \neq \emptyset$ for $i \in [n]$.

We now present the details of DCA. DCA is conceptually simple and easy to implement. Its main idea can be described as follows: relax the nested bound constraints (1c) first and solve the problem to optimality; if the obtained optimal solution satisfies the nested bound constraints, then this solution is optimal for DRAP-NC and we stop; otherwise find out the bound constraint that has the maximum violation, fix it at equality, and divide the original problem into two DRAP-NCs of smaller size and solve each subproblem recursively. Our algorithm combines three ingredients: (1) a recursive divide-and-conquer framework; (2) a scaling-based greedy algorithm for DRAP (Hochbaum 1994); (3) constraint relaxation and the use of constraint violation information to guide how to divide the problem.

For the ease of exposition, we introduce four dummy parameters a_0, b_0, a_n, and b_n, and set $a_0 = b_0 = 0$ and $a_n = b_n = B$. For integers i and j with $i \leq j$, we use $[i:j]$ to denote the set of integers $\{i, i+1, \ldots, j\}$. We define the problem DRAP-NC(s, e) below to be DRAP-NC with respect to activity $s, s+1, \ldots, e$ for $s, e \in [n]$.

$$\min_x \quad \sum_{i=s}^{e} f_i(x_i) \tag{2a}$$

$$\text{s.t.} \quad \sum_{i=s}^{e} x_i = b_e - a_{s-1}, \tag{2b}$$

$$a_i - a_{s-1} \leq \sum_{k=s}^{i} x_k \leq b_i - b_{s-1}, \qquad \forall i \in [s:e], \tag{2c}$$

$$0 \leq x_i \leq d_i, x_i \in \mathbb{Z}, \qquad \forall i \in [s:e], \tag{2d}$$

where we assume that $a_{s-1} = b_{s-1}$ and $a_e = b_e$. Moreover, we define the problem DRAP(s, e) to be a relaxation of DRAP-NC(s, e) by dropping the nested constraints (2c).

$$\min_x \quad \sum_{i=s}^{e} f_i(x_i) \tag{3a}$$

$$\text{s.t.} \quad \sum_{i=s}^{e} x_i = b_e - a_{s-1}, \tag{3b}$$

$$0 \le x_i \le d_i, x_i \in \mathbb{Z}, \qquad \forall i \in [s:e]. \tag{3c}$$

Our goal is to solve DRAP-NC$(1,n)$. To solve DRAP-NC(s,e) for any $s,e \in [n]$, we first solve the relaxation DRAP(s,e) to optimality. If the resulting optimal solution satisfies the nested constraints (2c), then we stop and output it as the optimal solution of DRAP-NC(s,e). Otherwise, we select an index K of a mostly violated nested constraint and fix the total resources consumed by the first K activities to be a_K or b_K, depending on which side of the nested constraints is violated. In particular, let $\bar{x}$ be an optimal solution of DRAP(s,e), then $K \in \arg\max_{i\in[s:e]} \left\{(a_i - a_{s-1}) - \sum_{k=s}^{i} \bar{x}_k, \sum_{k=s}^{i} \bar{x}_k - (b_i - b_{s-1})\right\}$. If $\sum_{k=s}^{K} \bar{x}_k > b_K - a_{s-1}$, then we set the constraint $\sum_{k=s}^{K} x_k \le b_K - a_{s-1}$ to equality for the rest of the algorithm; otherwise we have $\sum_{k=s}^{K} \bar{x}_k < a_K - a_{s-1}$ and set the constraint $\sum_{k=s}^{K} x_k \ge a_K - a_{s-1}$ to equality for the rest of the algorithm. With the above adjustment, we are able to divide DRAP-NC(s,e) into two subproblems DRAP-NC(s,K) and DRAP-NC$(K+1,e)$, and solve each separately. Finally, we combine the optimal solutions of DRAP-NC(s,K) and DRAP-NC$(K+1,e)$ to obtain an optimal solution of DRAP-NC(s,e). The subproblems DRAP-NC(s,K) and DRAP-NC$(K+1,e)$ will be solved recursively following the above procedure. The details of the algorithm are described in Algorithm 1.

We state our main result below.

THEOREM 1. *Algorithm 1 returns a global optimal solution of DRAP-NC at termination.*

Theorem 1 follows directly from Theorem 2 below.

THEOREM 2. *Let $(\bar{x}_s, \bar{x}_{s+1}, \ldots, \bar{x}_e)$ be an optimal solution of DRAP(s,e). Suppose $(\bar{x}_s, \bar{x}_{s+1}, \ldots, \bar{x}_e)$ violates at least one nested bound constraint of DRAP-NC(s,e) and the index of a mostly violated nested constraint is K. Then there exists an optimal solution $(\hat{x}_s, \hat{x}_{s+1}, \ldots, \hat{x}_e)$ of DRAP-NC(s,e) such that $\sum_{k=s}^{K} \hat{x}_k = a_K - a_{s-1}$ if $\sum_{i=s}^{K} \bar{x}_i < a_K - a_{s-1}$ or $\sum_{k=s}^{K} \hat{x}_k = b_K - a_{s-1}$ if $\sum_{i=s}^{K} \bar{x}_i > b_K - a_{s-1}$.*

We postpone the proof of Theorem 2 to Section 4, and first elaborate on Step 6 of Algorithm 1, i.e., how to solve the relaxation problem DRAP(s,e) to optimality.

Algorithm 1 DCA: a recursive algorithm for DRAP-NC(s,e)

```
1: **Input:** nested bounds $a_i$ and $b_i$ for $i \in [s-1:e]$ with $a_{s-1} = b_{s-1}$ and $a_e = b_e$; variable upper bound $d_i$ and function oracle $f_i$ for $i \in [s:e]$.
2: **Output:** an optimal solution $(\hat{x}_s, \ldots, \hat{x}_e)$ of DRAP-NC$(s,e)$.
3: **function** DRAP-NC$(s,e)$
4:     **if** $e = s$ **then return** $x_s = a_s - a_{s-1}$
5:     **end if**
6:     Solve DRAP$(s,e)$ and obtain an optimal solution $(\bar{x}_s, \ldots, \bar{x}_e)$
7:     **if** $(\bar{x}_s, \ldots, \bar{x}_e)$ satisfies the nested lower and upper bound constraints **then return** $(\bar{x}_s, \ldots, \bar{x}_e)$
8:     **end if**
9:     Find index $K$ of a mostly violated nested constraint, with the tie-breaking rule of choosing the maximum index.
10:    **if** $\sum_{k=s}^{K} \bar{x}_k > b_K - a_{s-1}$ **then** $a_K \leftarrow b_K$        ▷ Update bounds for subproblems
11:    **else** $b_K \leftarrow a_K$
12:    **end if**
13:    $(\hat{x}_s, \ldots, \hat{x}_K) \leftarrow$ DRAP-NC$(s,K)$
14:    $(\hat{x}_{K+1}, \ldots, \hat{x}_e) \leftarrow$ DRAP-NC$(K+1,e)$
15:    **return** $(\hat{x}_s, \ldots, \hat{x}_e)$
16: **end function**
```

3.1. Step 6 of Algorithm 1

At Step 6 of Algorithm 1, we need to solve a problem DRAP(s,e), which is an instance of DRAP with $e-s+1$ variables. As we mentioned in Section 2, the current fastest algorithm for DRAP runs in $\Theta(\max\{n \log \frac{B}{n}, n\})$ time (Frederickson and Johnson 1982, Hochbaum 1994). Here we use Hochbaum's algorithm to solve DRAP(s,e), since it is much easier to implement than that of Frederickson and Johnson (1982). Hochbaum's algorithm, which was called Algorithm GAP in Hochbaum (1994), was developed for a more general resource allocation problem with submodular constraints. It makes multiple calls to a greedy subroutine (called *greedy*(s) in Hochbaum (1994)) that returns a solution which uses as many resources as possible. To give a complete description of our algorithm, we present Hochbaum's algorithm specialized to DRAP with our own notations. The details of solving DRAP(s,e) are described in Algorithm 2 (corresponding to Algorithm GAP in Hochbaum (1994)) and Algorithm 3 (corresponding to *greedy*(s) in Hochbaum (1994)). We use $\boldsymbol{e}$, $\mathbf{0}$, and $\boldsymbol{e}^i$ to denote a column vector of all ones, a column vector of all zeros, and a unit vector with the i-th entry being one, respectively.

Algorithm 2 An algorithm for DRAP(s,e) (Hochbaum 1994)

```
1: **Input:** variable upper bound $d_i$ and value oracle $f_i$ for $i \in [s:e]$ and the total amount
   of resources $B = b_e - a_{s-1}$.
2: **Output:** an optimal solution $x^*$ of DRAP$(s,e)$.
3: **function** DRAP$(s,e)$
4:     **Initialization:** $\delta \leftarrow \left\lceil \frac{B}{2n} \right\rceil, x \leftarrow \mathbf{0}$
5:     **while** $\delta > 1$ **do**
6:         $x \leftarrow$ Greedy$(\delta, x, B)$
7:         $x \leftarrow \max\{x - \delta \boldsymbol{e}, \mathbf{0}\}$, $\delta \leftarrow \left\lceil \frac{\delta}{2} \right\rceil$
8:     **end while**
9:     $x^* \leftarrow$ Greedy$(1, x, B)$
10:    **return** $x^*$
11: **end function**
```

4. Proof of Theorem 2

In this section, we provide a complete proof of Theorem 2. The proof relies on the connection between the optimal solution of the discrete resource allocation problem and the optimal solution of its continuous relaxation through piecewise linear functions. This connection was also used in proving the correctness of MDA in Vidal et al. (2018). Given a convex function $f(x)$, we define a piecewise linear function as follows:

$$f^{PL}(x) = f(\lfloor x \rfloor) + (x - \lfloor x \rfloor) \times (f(\lceil x \rceil) - f(\lfloor x \rfloor)), \tag{4}$$

where $\lfloor x \rfloor$ and $\lceil x \rceil$ give the greatest integer less than or equal to x and the least integer greater than or equal to x, respectively. It can be easily verified that $f^{PL}(x)$ has the same values as $f(x)$ at integer points. Consider any instance of DRAP-NC(s,e). We define DRAP-NC$^{PL}(s,e)$ to be a problem by replacing the function f_i in DRAP-NC(s,e) with the function f_i^{PL} for each $i \in [s:e]$. Similarly, we define RAP-NC$^{PL}(s,e)$, DRAP$^{PL}(s,e)$,

Algorithm 3 The greedy subroutine Greedy(δ, y, B) (Hochbaum 1994)

```
1: Input: integer step size $\delta$, a vector $y$ that satisfies the variable bound constraints of DRAP$(s,e)$ (i.e., $y_i \in [0, d_i]$ for $i \in [s:e]$), the total amount of available resources $B = b_e - a_{s-1}$, and the value oracle $f_i$ for $i \in [s:e]$.
2: Output: a solution $x$ with the property that $x_i \in [0, d_i]$ for $i \in [s:e]$ and $\sum_{i=s}^{e} x_i \geq \sum_{i=s}^{e} y_i$.
3: function GREEDY$(\delta, y, B)$
4:     Initialization: $x \leftarrow y$, $R \leftarrow B - y^\top \mathbf{e}$, $E \leftarrow \{s, s+1, \ldots, e\}$
5:     while $R > 0$ and $E \neq \emptyset$, do
6:         Find $i \in E$ such that $\Delta_i(x_i) = \min_{j \in E}\{\Delta_j(x_j)\}$ where $\Delta_j(x_j) = f_j(x_j + 1) - f_j(x_j)$.    ▷ Find out the index with the smallest marginal cost.
7:         if $x_i + 1 > d_i$ then $E \leftarrow E \setminus \{i\}$    ▷ Check if the variable bound constraint is violated.
8:         else if $x_i + \delta > d_i$ or $\delta > R$ then
9:             $\delta' \leftarrow \min\{R, d_i - x_i\}$, $x_i \leftarrow x_i + \delta'$, $R \leftarrow R - \delta'$
10:            $E \leftarrow E \setminus \{i\}$
11:        else $x_i \leftarrow x_i + \delta$, $R \leftarrow R - \delta$
12:        end if
13:    end while
14:    return $x$
15: end function
```

and RAP$^{PL}(s,e)$ to be the continuous relaxation of DRAP-NC$^{PL}(s,e)$, DRAP-NC$^{PL}(s,e)$ without nested bound constraints, the continuous relaxation of DRAP-NC$^{PL}(s,e)$ without nested bound constraints, respectively. We first introduce several propositions needed later for the proof.

PROPOSITION 1. *Any solution of DRAP-NC(s,e) is optimal if and only if it is an optimal solution of DRAP-NC$^{PL}(s,e)$. Any solution of DRAP(s,e) is optimal if and only if it is an optimal solution of DRAP$^{PL}(s,e)$.*

Proof. DRAP-NC(s,e) and DRAP-NC$^{PL}(s,e)$ have the same set of feasible solutions. Each feasible solution has the same objective value in DRAP-NC(s,e) as in

DRAP-NC$^{PL}(s,e)$, since f_i and f_i^{PL} coincide in the integer domain for each i. Therefore, any solution of DRAP-NC(s,e) is optimal if and only if it is an optimal solution of DRAP-NC$^{PL}(s,e)$. The second statement follows directly from the fact that DRAP(s,e) and DRAP$^{PL}(s,e)$ are special cases of DRAP-NC(s,e) and DRAP-NC$^{PL}(s,e)$ respectively by setting $a_i = -\infty$ and $b_i = \infty$ for each i. □

Proposition 2. *(Vidal et al. 2018, Theorem 4) Any optimal solution of DRAP-NC$^{PL}(s,e)$ is also an optimal solution of its continuous relaxation RAP-NC$^{PL}(s,e)$.*

Proposition 3. *Any optimal solution of DRAP(s,e) is an optimal solution of RAP$^{PL}(s,e)$.*

Proof. By Proposition 1, any optimal solution of DRAP(s,e) is an optimal solution of DRAP$^{PL}(s,e)$. By Proposition 2, any optimal solution of DRAP$^{PL}(s,e)$ is an optimal solution of RAP$^{PL}(s,e)$ by setting $a_i = -\infty$ and $b_i = \infty$ for each i. Then the result follows directly. □

With Proposition 3, we are now ready to prove Theorem 2.

Proof of Theorem 2 We will prove that there exists an optimal solution $(\hat{x}_s, \ldots, \hat{x}_e)$ of DRAP-NC(s,e) satisfying $\sum_{k=s}^{K} \hat{x}_k = b_K - a_{s-1}$ if $\sum_{k=s}^{K} \bar{x}_k > b_K - a_{s-1}$. The result for the other case in which $\sum_{k=s}^{K} \bar{x}_k < a_K - a_{s-1}$ can be proven analogously.

We prove the result by contradiction. Suppose that there does not exist any optimal solution of DRAP-NC(s,e) such that the constraint $\sum_{k=s}^{K} x_k \le b_K - a_{s-1}$ is satisfied at equality. Since any instance of DRAP-NC(s,e) has only a finite number of optimal solutions, we select the optimal solution that maximizes $\sum_{k=s}^{K} x_k$ among all optimal solutions. Call this solution $\hat{x} = (\hat{x}_s, \ldots, \hat{x}_e)$ and we have $\sum_{k=s}^{K} \hat{x}_k < b_K - a_{s-1}$. We will show that we could create another optimal solution $\tilde{x}$ of DRAP-NC(s,e) such that $\sum_{k=s}^{K} \tilde{x}_k > \sum_{k=s}^{K} \hat{x}_k$, contradicting to the choice of $\hat{x}$.

The construction works as follows. Let I be the maximum element in the set $\{i \mid \sum_{k=s}^{i} \hat{x}_k = b_i - a_{s-1}, s-1 \le i < K\}$ and J be the minimum element in the set $\{i \mid \sum_{k=s}^{i} \hat{x}_k = b_i - a_{s-1}, K < i \le e\}$. Both I and J are well-defined since the sets defining them contain $s-1$ and e respectively. We claim that there exist integer $p \in [I+1 : K]$ and $q \in [K+1 : J]$ such that $\bar{x}_p > \hat{x}_p$ and $\bar{x}_q < \hat{x}_q$. To see this, since K is the index of a nested constraint with the largest violation, $\sum_{k=s}^{K} \bar{x}_k - b_K \ge \sum_{k=s}^{I} \bar{x}_k - b_I$. Then $\sum_{k=s}^{K} \bar{x}_k - \sum_{k=s}^{I} \bar{x}_k \ge b_K - b_I$. Meanwhile, $\sum_{k=s}^{K} \hat{x}_k < b_K - a_{s-1}$ and $\sum_{k=s}^{I} \hat{x}_k = b_I - a_{s-1}$. Thus $\sum_{k=s}^{K} \bar{x}_k - \sum_{k=s}^{I} \bar{x}_k \ge b_K - b_I >$

$(\sum_{k=s}^{K}\hat{x}_k + a_{s-1}) - b_I = \sum_{k=s}^{K}\hat{x}_k - \sum_{k=s}^{I}\hat{x}_k$. Therefore $\sum_{k=I+1}^{K}\bar{x}_k > \sum_{k=I+1}^{K}\hat{x}_k$. Then there must exist some $p \in [I+1:K]$ such that $\bar{x}_p > \hat{x}_p$. Similarly, we can find some $q \in [K+1:J]$ such that $\bar{x}_q < \hat{x}_q$.

We claim that for such p and q, the following inequalities hold.

$$\sup \partial f_p^{PL}(\hat{x}_p) \le \inf \partial f_p^{PL}(\bar{x}_p) \le \sup \partial f_q^{PL}(\bar{x}_q) \le \inf \partial f_q^{PL}(\hat{x}_q). \tag{5}$$

To see this, since the solution $\bar{x}$ is an optimal solution of DRAP(s,e), by Proposition 3, it is also an optimal solution of the continuous optimization problem RAP$^{PL}(s,e)$. In addition, $\bar{x}_p > \hat{x}_p \ge 0$ and $\bar{x}_q < \hat{x}_q \le d_q$. Therefore, the solution $\bar{x}$ should satisfy the Karush-Kuhn-Tucker conditions for RAP$^{PL}(s,e)$ (see Chapters 28-30 of Rockafellar (1970) for Karush-Kuhn-Tucker conditions with subdifferentials for optimization problems involving non-smooth functions). In particular, there must exist some dual variable λ such that

$$\inf \partial f_p^{PL}(\bar{x}_p) \le \lambda \le \sup \partial f_q^{PL}(\bar{x}_q),$$

where $\partial f_i^{PL}(x)$ is the subdifferential of $f_i^{PL}(x)$ for $i \in [s:e]$. Then by the monotonicity of the subdifferential of convex functions, we prove (5).

Now we create an integer vector $\tilde{x} = (\tilde{x}_s, \ldots, \tilde{x}_e)$ such that $\tilde{x}_p = \hat{x}_p + 1$, $\tilde{x}_q = \hat{x}_q - 1$, and $\tilde{x}_i = \hat{x}_i$ otherwise. We claim that $\tilde{x}$ is also an optimal solution of DRAP-NC(s,e). To check the feasibility of $\tilde{x}$, note that $\hat{x}_p < \bar{x}_p \le d_p$ and $\hat{x}_p$ and $\bar{x}_p$ are integers, so $\tilde{x}_p \le d_p$. Similarly, we can show that $\tilde{x}_q \ge 0$. To check that $\tilde{x}$ satisfies all nested constraints, note that by the choice of p and q, $\sum_{k=s}^{j}\hat{x}_k < b_j - a_{s-1}$ for each $j \in [p:q]$. Since all parameters are integral, for each $j \in [p:q]$, $\sum_{k=s}^{j}\tilde{x}_k \le b_j - a_{s-1}$. To check the optimality of $\tilde{x}$, note that f_p^{PL} and f_q^{PL} are both piecewise linear functions with break points only at integer points. Then from (5) we have $f_p^{PL}(\tilde{x}_p) + f_q^{PL}(\tilde{x}_q) \le f_p^{PL}(\hat{x}_p) + f_q^{PL}(\hat{x}_q)$. But $\sum_{k=s}^{K}\tilde{x}_k = \sum_{k=s}^{K}\hat{x}_k + 1$, contradicting to the choice of $\hat{x}$ such that it maximizes $\sum_{k=s}^{K}x_k$ among all optimal solutions. □

5. Time complexity of Algorithm 1

We first introduce the notations of O, Ω, and Θ when the functions involved have two variables (Cormen et al. 2009).

Definition 1.

1. Given two functions $f(n,B)$ and $g(n,B)$, $f(n,B) = O(g(n,B))$ if there exist positive real numbers K and M such that

$$0 \le f(n,B) \le Kg(n,B) \text{ for all } n \ge M \text{ or } B \ge M.$$

2. Given two functions $f(n,B)$ and $g(n,B)$, $f(n,B)=\Omega(g(n,B))$ if there exist positive real numbers K and M such that

$$0 \le Kg(n,B) \le f(n,B) \text{ for all } n \ge M \text{ or } B \ge M.$$

3. Given two functions $f(n,B)$ and $g(n,B)$, $f(n,B)=\Theta(g(n,B))$ if there exist positive real numbers K_1, K_2, and M such that

$$0 \le K_1 g(n,B) \le f(n,B) \le K_2 g(n,B) \text{ for all } n \ge M \text{ or } B \ge M.$$

From the definitions above, it is not difficult to verify that $f(n,B)=\Theta(g(n,B))$ if and only if $f(n,B)=O(g(n,B))$ and $f(n,B)=\Omega(g(n,B))$.

The upper bound on the running time of the algorithm is shown by the Proposition below.

Proposition 4. *The worst-case time complexity of Algorithm 1 is* $O(\max\{n^2\log\frac{B}{n}, n^2\})$.

Proof. We will show that the recursion tree of Algorithm 1 has a maximum depth $n-2$ and the running time at each level of the recursion tree is $O(\max\{n\log\frac{B}{n}, n\})$.

At each level of the recursion tree, Algorithm 1 fixes at least one nested bound constraint (out of a pair of lower bound and upper bound constraints) at equality. DRAP-NC$(1,n)$ has $n-1$ pairs of nested bound constraints, so the recursion tree of the algorithm has a maximum depth of $n-2$.

We now show that the running time at each level of the recursion tree is $O(\max\{n\log\frac{B}{n}, n\})$. Without loss of generality, we assume that the base of the logarithm in the complexity result is 2 in the rest of the proof. Consider one level of the recursion tree. Suppose that there are m DRAP-NC subproblems at this level and the ith DRAP-NC subproblem has n_i variables and B_i units of resources for $i \in [m]$. We have

$$\sum_{i=1}^{m} n_i = n \text{ and } \sum_{i=1}^{m} B_i = B.$$

For the ith DRAP-NC subproblem, it takes at most $K_1\max\{n_i\log\frac{B_i}{n_i}, n_i\}$ time for Hochbaum's greedy subroutine to solve the DRAP relaxation for some positive integers K_1. In addition, it takes at most $K_2 n_i$ time to find out the mostly violated nested bound constraint for some positive integer K_2. Therefore the total running time of DCA at this level of recursion tree is at most $\sum_{i=1}^m K_1 \max\{n_i \log \frac{B_i}{n_i}, n_i\} + \sum_{i=1}^m K_2 n_i$. We have

$$
\begin{aligned}
&\sum_{i=1}^m K_1 \max\{n_i \log \frac{B_i}{n_i}, n_i\} + \sum_{i=1}^m K_2 n_i \\
&= \sum_{i=1}^m K_1 n_i \log(\max\{\frac{B_i}{n_i}, 2\}) + \sum_{i=1}^m K_2 n_i \\
&= K_1 \log \prod_{i=1}^m \left(\max\{\frac{B_i}{n_i}, 2\}\right)^{n_i} + K_2 n \\
&\le K_1 \log \left(\frac{\sum_{i=1}^m \left(\max\{\frac{B_i}{n_i}, 2\} \times n_i\right)}{\sum_{i=1}^m n_i}\right)^{\sum_{i=1}^m n_i} + K_2 n \\
&= K_1 \log \left(\frac{\sum_{i=1}^m (\max\{B_i, 2n_i\})}{\sum_{i=1}^m n_i}\right)^{\sum_{i=1}^m n_i} + K_2 n \\
&\le K_1 \log \left(\frac{\sum_{i=1}^m (B_i + 2n_i)}{\sum_{i=1}^m n_i}\right)^{\sum_{i=1}^m n_i} + K_2 n \\
&= K_1 \log \left(\frac{B}{n} + 2\right)^n + K_2 n \\
&\le K_1 \log \left(2 \max\{\frac{B}{n}, 2\}\right)^n + K_2 n \\
&= K_1 n (\log(\max\{\frac{B}{n}, 2\}) + 1) + K_2 n \\
&= K_1 n \log(\max\{\frac{B}{n}, 2\}) + (K_1 + K_2) n \\
&\le (2K_1 + K_2) n \log(\max\{\frac{B}{n}, 2\}) \\
&= (2K_1 + K_2) \max\{n \log \frac{B}{n}, n\} \\
&= O\left(\max\{n \log \frac{B}{n}, n\}\right),
\end{aligned}
$$

where the first inequality follows from the fact that the geometric mean of the $\sum_{i=1}^m n_i$ positive numbers is smaller than or equal to their arithmetic mean. $\square$

Proposition 4 gives an upper bound on the worst-case running time of our algorithm. The following instance shows that that upper bound is indeed tight. This instance is a minor modification of an instance proposed by Thibaut Vidal (Vidal 2018) for an earlier version of our algorithm, in which we did not assume $a_i < b_i$ for each i. Consider the following instance of DRAP-NC:

$$\min \quad \sum_{i=1}^{n} x_i^2 \tag{6a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_i = (-1)^n n, \tag{6b}$$

$$(-1)^i i \leq \sum_{k=1}^{i} x_k \leq (-1)^i i + 1, \qquad \forall i \in [n-1], \tag{6c}$$

$$-2n \leq x_i \leq 2n, x_i \in \mathbb{Z}, \qquad \forall i \in [n]. \tag{6d}$$

This instance can be transformed into Formulation (1) through variable substitution. This instance is feasible since it has a solution $(-1, 3, -5, \ldots, (-1)^n(2n-1))$.

We claim that DCA needs $\Omega(n)$ recursions to terminate for this instance. To see this, in the first iteration of DCA, the nested constraints (6c) are relaxed and we solve DRAP$(1, n)$ below (assuming n to be even here):

$$\min \quad \sum_{i=1}^{n} x_i^2 \tag{7a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_i = n, \tag{7b}$$

$$-2n \leq x_i \leq 2n, x_i \in \mathbb{Z}, \qquad \forall i \in [n]. \tag{7c}$$

The optimal solution is $x_1^* = \ldots = x_n^* = 1$. The mostly violated nested constraint under this optimal solution is the $(n-1)$-th upper-bound constraint $\sum_{k=1}^{n-1} x_k \leq -(n-1)+1$, which has a violation of $(n-1) - [-(n-1)+1] = 2n-3$. Therefore, the constraint $\sum_{k=1}^{n-1} x_k \leq -(n-1)+1 = -n+2$ is set at equality for the rest of the algorithm, and DRAP$(1, n)$ is divided into DRAP-NC$(1, n-1)$ and DRAP-NC(n, n).

Since $\sum_{k=1}^{n-1} x_k = -n+2$ and $\sum_{k=1}^{n} x_k = n$, the problem DRAP-NC(n, n) is as follows:

$$\min \quad x_n^2 \tag{8a}$$

$$\text{s.t.} \quad x_n = 2n-2, \tag{8b}$$

$$-2n \leq x_n \leq 2n, x_n \in \mathbb{Z}. \tag{8c}$$

It has a trivial optimal solution $x_n^* = 2n-2$.

The problem DRAP-NC$(1, n-1)$ is reduced to the following:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n-1} x_i^2 && && \text{(9a)}\\
\text{s.t.} \quad & \sum_{k=1}^{n-1} x_k = -n+2, && && \text{(9b)}\\
& (-1)^i i \le \sum_{k=1}^{i} x_k \le (-1)^i i + 1, && \forall i \in [n-2], && \text{(9c)}\\
& -2n \le x_i \le 2n, x_i \in \mathbb{Z}, && \forall i \in [n-1]. && \text{(9d)}
\end{aligned}$$

We solve DRAP-NC$(1, n-1)$ similarly by first relaxing the nested constraints. An optimal solution for its relaxation DRAP$(1, n-1)$ is $x_1^* = \ldots = x_{n-2}^* = -1$ and $x_{n-1}^* = 0$. The mostly violated nested constraint in (9c) under this optimal solution is the $(n-2)$-th lower bound constraint $n-2 \le \sum_{k=1}^{n-2} x_k$, which has a violation of $(n-2) - [-(n-2)] = 2n-4$. Therefore, the constraint $n-2 \le \sum_{k=1}^{n-2} x_k$ is set at equality for the rest of the algorithm, and the problem DRAP-NC$(1, n-1)$ is divided into DRAP-NC$(1, n-2)$ and DRAP-NC$(n-1, n-1)$.

The problem DRAP-NC$(n-1, n-1)$ has a trivial optimal solution. The problem DRAP-NC$(1, n-2)$ is reduced to the following:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n-2} x_i^2 && && \text{(10a)}\\
\text{s.t.} \quad & \sum_{k=1}^{n-2} x_k = n-2, && && \text{(10b)}\\
& (-1)^i i \le \sum_{k=1}^{i} x_k \le (-1)^i i + 1, && \forall i \in [n-3], && \text{(10c)}\\
& -2n \le x_i \le 2n, x_i \in \mathbb{Z}, && \forall i \in [n-2]. && \text{(10d)}
\end{aligned}$$

DRAP-NC$(1, n-2)$ has the same structure and coefficients as DRAP-NC$(1, n)$ with two less variables and nested bound constraints. It could be solved recursively as discussed before. Thus DCA only fixes one variable at each level of the recursion tree, and the depth of the recursion tree will be $\Omega(n)$ for this instance.

Now we look at the running time of Algorithm 2 in solving the relaxation DRAP$(1, n)$ with $B = 2n^2 + n$. Algorithm 2 needs to run $\Omega(\log \frac{B}{n})$ iterations of $\Theta(n)$ operations at

Line 7, so its running time is $\Omega(n\log\frac{B}{n})$. Therefore the overall running time of DCA for this instance will be $\Omega(n^2\log\frac{B}{n}) = \Omega(\max\{n^2\log\frac{B}{n}, n^2\})$. Combining this fact with Proposition 4, we have the following result.

THEOREM 3. *The worst-case time complexity of Algorithm 1 is* $\Theta(\max\{n^2\log\frac{B}{n}, n^2\})$.

We would like to point out that the analysis above is based on the worst-case scenario, which means the estimation that the running time of DCA grows proportionally with the maximum of $n^2\log\frac{B}{n}$ and n^2 may be too conservative. As will be shown by the analysis in the rest of this section and the computational experiments in Section 6, our algorithm could be much more efficient than this asymptotic upper bound.

5.1. An example beyond the worst-case analysis

The previous section shows that the worst-case time complexity of DCA is $\Theta(\max\{n^2\log\frac{B}{n}, n^2\})$, which is much worse than the time complexity $O(n\log n\log B)$ of MDA (Vidal et al. 2018). This comparison, however, does not tell the full story of the efficiency of the two algorithms. The running time of DCA is much more dependent on the instance, similar to the fact that the actual running time of a branch-and-bound algorithm depends heavily on the particular integer program it solves. DCA could be very efficient for some instances, as illustrated below.

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n+1} x_i^2 - Mx_{n+1} && && \text{(11a)}\\
\text{s.t.} \quad & \sum_{i=1}^{n+1} x_i = n, && && \text{(11b)}\\
& i \le \sum_{k=1}^{i} x_k \le i+1, && \forall i \in [n], && \text{(11c)}\\
& 0 \le x_i \le 2n, x_i \in \mathbb{Z}, && \forall i \in [n+1]. && \text{(11d)}
\end{aligned}$$

The constant M in the objective function is sufficiently large such that the optimal solution of the problem with the objective (11a) and constraints (11b) and (11d) is $x_1^* = \ldots = x_n^* = 0$ and $x_{n+1}^* = n$. This could be achieved, for example, by setting $M = 100n^2$. To solve this instance, DCA only needs two recursions and solves three DRAP subproblems; on the other hand, MDA needs $1 + \lceil\log n\rceil$ recursions and solves $\Theta(n)$ DRAP subproblems.

To see this, by slightly abusing the notation, we call problem (11) DRAP-NC$(1, n+1)$. Observe that it has a unique optimal solution $x_1^* = \ldots = x_n^* = 1$ and $x_{n+1}^* = 0$. When applying DCA to this instance, we first solve its relaxation without the nested constraints (11c), which we called DRAP$(1, n+1)$. The optimal solution is $x_1^* = \ldots = x_n^* = 0$ and $x_{n+1}^* = n$. The nested constraint $n \leq \sum_{k=1}^n x_k$ has the largest violation under this optimal solution. Then we set $\sum_{k=1}^n x_k = n$ for the rest of the algorithm, and divide the problem into two subproblems DRAP-NC$(1, n)$ and DRAP-NC$(n+1, n+1)$. The two subproblems are described as follows.

DRAP-NC$(1, n)$:

$$\min \quad \sum_{i=1}^n x_i^2 \tag{12a}$$

$$\text{s.t.} \quad \sum_{k=1}^n x_k = n, \tag{12b}$$

$$i \leq \sum_{k=1}^i x_k \leq i+1, \qquad \forall i \in [n-1], \tag{12c}$$

$$0 \leq x_i \leq 2n, x_i \in \mathbb{Z}, \qquad \forall i \in [n]. \tag{12d}$$

DRAP-NC$(n+1, n+1)$:

$$\min \quad x_{n+1}^2 - M x_{n+1} \tag{13a}$$

$$\text{s.t.} \quad x_{n+1} = 0, \tag{13b}$$

$$0 \leq x_{n+1} \leq 2n, x_{n+1} \in \mathbb{Z}. \tag{13c}$$

The subproblem DRAP-NC$(n+1, n+1)$ has a trivial optimal solution $x_{n+1}^* = 0$. On the other hand, to solve the subproblem DRAP-NC$(1, n)$, DCA first solves its relaxation without the nested constraints, DRAP$(1, n)$. Observe that DRAP$(1, n)$ has a unique optimal solution $x_1^* = \ldots = x_n^* = 1$, which does not violated any nested constraint. Therefore, we obtain the optimal solution to the original problem DRAP-NC$(1, n+1)$ by solving three DRAP subproblems in two recursions.

In comparison, MDA needs to solve $\Theta(n)$ DRAP subproblems at $1 + \lceil \log n \rceil$ recursions to solve DRAP-NC$(1, n+1)$. To see this, at each recursion, MDA breaks down a DRAP-NC instance into two subproblems of equal size, and each subproblem is further divided recursively until the subproblem has exactly one variable, which is then solved trivially.

Using optimal solutions of the two subproblems of smaller sizes, MDA generates stronger variable bounds for DRAP-NC subproblems of larger sizes (at one level higher in the recursion tree). Vidal et al. (2018) showed that the nested bound constraints become redundant with the new variable bounds and each DRAP-NC subproblem can be reduced to solving four DRAP problems. Since MDA always divides the problem evenly, the depth of the recursion tree for an instance with n variables will be $1 + \lceil \log n \rceil$ and the total number of calls to the DRAP subroutine will be $\Theta(n)$.

We hope that the instance above could provide some insight on why the worst-case analysis sometimes overestimates the running time of DCA: DCA only divides the problem when necessary, i.e., when the intermediate solution found violates some nested bound constraint. The number of recursions may be significantly fewer than $\Omega(n)$ in the worst case.

6. Numerical experiments

In this section, we evaluate the performance of DCA through two sets of experiments involving DRAP-NC instances with different types of convex objectives. The first experiment is a sanity test on the correctness of DCA. We apply DCA to a set of DRAP-NC instances with linear objectives, and compare its output with the output of a state-of-the-art optimization solver Gurobi (Gurobi Optimization 2019). For all the test instances we generate, the two approaches always produce the same optimal solutions. In the second experiment, we apply DCA to solve DRAP-NC instances with three classes of nonlinear objectives motivated by applications like project crashing (Foldes and Soumis 1993) and vessel fuel minimization (Ronen 1982). We compare the results of DCA with those of MDA. Note that in Vidal et al. (2018), MDA was only implemented for continuous RAP-NC instances, and bisection search was used to solve the RAP relaxation. For MDA to solve DRAP-NC, DRAP relaxations need to be solved. For a fair comparison with DCA, we implement MDA on our own, replacing bisection search with Hochaum's scaling-based algorithm for the DRAP subproblem. All programs are coded in Java and the experiments are conducted on a desktop computer with an AMD 3700X CPU, 32 gigabyte memory, a Windows 10 operating system, and Gurobi 9.0.2. The time limit is set to 1200 seconds for each instance. To accurately report the solution time for instances that are solved in less than one millisecond, we solve each each such instance 100 times and report the average running time. All the code related to the implementation of our algorithm and numerical experiments are available at an online repository `https://github.com/qqqhe/dca`.

6.1. Test instance generation

The parameters in the constraints (1b) to (1d) of all test instances are generated in the same way as follows.

- The variable upper bound d_i's are drawn from a discrete uniform distribution over the set $\{1, 2, \ldots, V_b\}$, where V_b is a parameter we could change.
- The bounds a_i's and b_i's in the nested constraints are generated following a similar procedure of generating RAP-NC instances in Vidal et al. (2018), in order to guarantee the feasibility of an instance. In particular, we first generate two sequences of integers $\{v_i\}_{i=0}^n$ and $\{w_i\}_{i=0}^n$ as follows.

$$v_0 = w_0 = 0,$$
$$v_i = v_{i-1} + X_i^v,$$
$$w_i = w_{i-1} + X_i^w,$$

where X_i^w and X_i^v are drawn independently from a discrete uniform distribution over the set $[0 : d_i]$. Then set $a_i = \min\{v_i, w_i\}$ and $b_i = \max\{v_i, w_i\}$ for $i \in [n]$. Finally, we set $B = \max\{a_n, b_n\}$.

By changing the values of the pair (n, V_b), we are able to generate DRAP-NC instances of different sizes. As for how the objective function is generated for each instance, we will defer the details to each of the sections below.

6.2. DRAP-NC with linear objectives

In the first experiment, we set the function $f_i(x)$ in (1a) to be $f_i(x) = p_i x$, where p_i is drawn from a uniform distribution over $[-1, 1]$. We use the procedure described in Section 6.1 to generate constraints of the test instances. We choose $n = 2^i \times 100$ for $i \in [5 : 15]$ and $V_b \in \{10, 100\}$. For each (n, V_b) pair, we generate 10 instances, so there are $11 \times 2 \times 10 = 220$ test instances for the first experiment.

We apply DCA and Gurobi to solve each instance, and compare their optimal solutions. Normally Gurobi will have a difficult time in solving mixed-integer linear programs with millions of integer variables, but in this case we can use Gurobi's linear programming (LP) solver. The reason is that the coefficient matrix in the constraints of DRAP-NC is totally unimodular, so DRAP-NC with a linear objective can be solved as a linear program by ignoring the integrality constraints (1e). This greatly reduces the solution time of Gurobi.

We also notice that the following formulation trick further reduces the solution time of Gurobi's LP solver significantly. By introducing new variables $y_i = \sum_{k=1}^{i} x_k - a_i$ for $i \in [n-1]$, we could reformulate DRAP-NC as follows:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} f_i(x_i) \\ \text{s.t.} \quad & x_n + y_{n-1} = B - a_{n-1}, \\ & x_1 - y_1 = a_1, \\ & x_i + y_{i-1} - y_i = a_i - a_{i-1}, && \forall i \in [2:n-1], \\ & 0 \le y_i \le b_i - a_i, && \forall i \in [n-1], \\ & 0 \le x_i \le d_i, && \forall i \in [n]. \end{aligned}$$

Such formulation has a more sparse coefficient matrix than that of the original formulation (1). We observe that Gurobi is on average thirty to one hundred times faster in solving the new formulation than the original formulation.

Both DCA and Gurobi solve the 220 test instances very efficiently, in less than a minute for each instance. The optimal solutions obtained by both approaches are exactly the same for each instance. This validates the correctness of the DCA. We also summarize the running time of both approaches in Table 1 and Figure 1. The running time is averaged over 10 instances for each (n, V_b) pair.

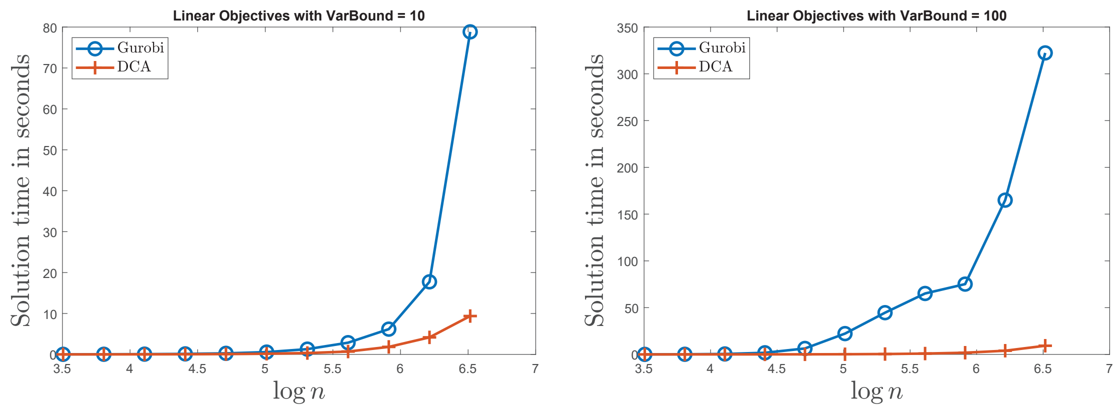

Figure 1 **Solution time of DCA and Gurobi for instances with linear objectives.**

Parameters		DCA	Gurobi	Parameters		DCA	Gurobi
n	V_b	Time (s)	Time (s)	n	V_b	Time (s)	Time (s)
3,200	10	0.006	0.023	3,200	100	0.004	0.045
6,400	10	0.008	0.036	6,400	100	0.010	0.144
12,800	10	0.019	0.066	12,800	100	0.016	0.479
25,600	10	0.035	0.128	25,600	100	0.038	1.738
51,200	10	0.082	0.261	51,200	100	0.092	6.365
102,400	10	0.164	0.564	102,400	100	0.182	22.246
204,800	10	0.331	1.288	204,800	100	0.402	44.614
409,600	10	0.697	2.866	409,600	100	0.903	65.143
819,200	10	1.836	6.175	819,200	100	1.803	75.158
1,638,400	10	4.164	17.736	1,638,400	100	3.940	164.917
3,276,800	10	9.375	78.826	3,276,800	100	9.263	322.344

Table 1 Solution statistics of DCA and Gurobi for instances with linear objectives.

It can be seen that the LP solver of Gurobi is able to solve instances with more than three million variables in less than six minutes, but DCA is on average 20 times faster. From Figure 1, we can see that the running time of DCA grows much slower than that of Gurobi with the problem size n.

6.3. DRAP-NC with nonlinear objectives

In the following, we test the performance of DCA on DRAP-NC instances with three classes of convex nonlinear objectives used in the computational experiments for MDA in Vidal et al. (2018): [F], [CRASH], and [FUEL].

$$[\mathrm{F}]: f_i(x) = \frac{x^4}{4} + p_i x, \tag{15}$$

$$[\mathrm{CRASH}]: f_i(x) = k_i + \frac{p_i}{x}, \tag{16}$$

$$[\mathrm{FUEL}]: f_i(x) = p_i \times c_i \times \left(\frac{c_i}{x}\right)^3. \tag{17}$$

We report two solution statistics of both DCA and MDA on these instances: running time and the total number of DRAP subproblems solved. We report the second statistics due to the fact that both DCA and MDA spend most of the solution time on solving DRAP subproblems. The total number of DRAP subproblems solved could shed some light on the different running time of both algorithms.

6.3.1. DRAP-NC with objectives [F] In each instance, the function $f_i(x)$ has the form $f_i(x) = \frac{x^4}{4} + p_i x$, where p_i is drawn from a uniform distribution over $[-1,1]$. We use the procedure described in Section 6.1 to generate constraints of the test instances. We fix the parameter V_b to be 100 and set $n = 2^i \times 100$ for $i \in [3:16]$. For each value of n, we generate 10 instances, so there are $14 \times 10 = 140$ instances. The average running time as well as the average number of DRAP subproblems solved are reported in Table 2 and Figure 2.

Parameters	Time (s)			Average number of DRAPs solved		
n	DCA	MDA	Ratio	DCA	MDA	Ratio
800	0.01	0.03	33%	107.2	6,396	1.68%
1,600	0.02	0.04	50%	115.4	12,796	0.90%
3,200	0.03	0.09	33%	146.6	25,596	0.57%
6,400	0.08	0.21	38%	308.2	51,196	0.60%
12,800	0.17	0.48	35%	305.4	102,396	0.30%
25,600	0.40	1.10	36%	373.0	204,796	0.18%
51,200	0.98	2.47	40%	807.8	409,596	0.20%
102,400	2.25	5.64	40%	966.8	819,196	0.12%
204,800	5.46	12.80	43%	1510.8	1,638,396	0.09%
409,600	16.84	29.01	58%	2903.6	3,276,796	0.09%
819,200	34.78	68.37	51%	3364.4	6,553,596	0.05%
1,638,400	91.65	164.43	56%	5411.6	13,107,196	0.04%
3,276,800	205.40	397.91	52%	5724.4	26,214,396	0.02%
6,553,600	535.65	964.45	56%	11250.4	52,428,796	0.02%

Table 2 Solution statistics of DCA and MDA for instances with objectives [F].

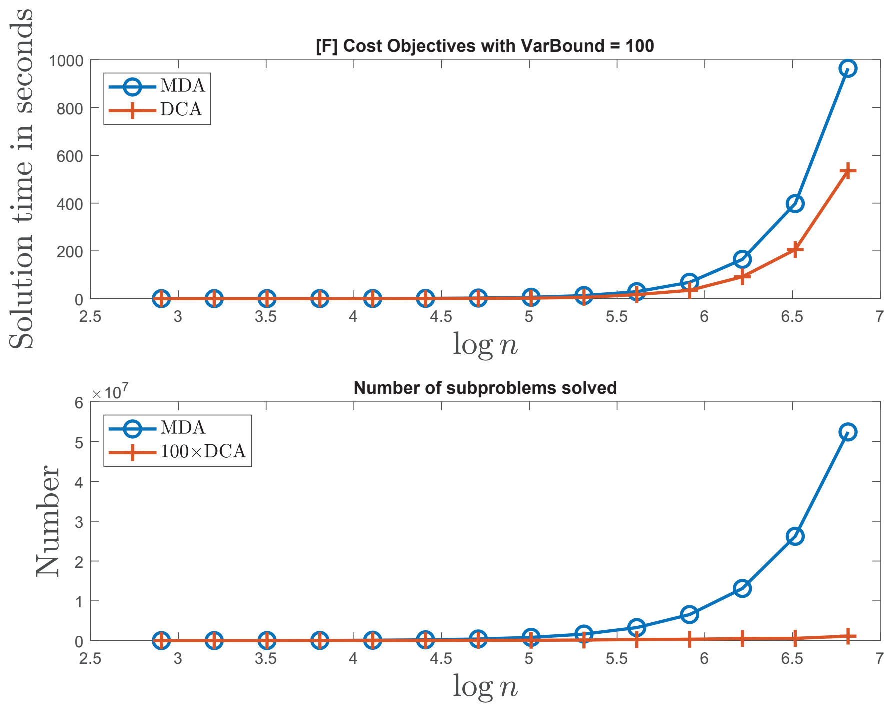

Figure 2 **The growth of running time and the number of DRAP subproblems solved for instances with objectives [F].**

It can be seen that the average running time DCA is no more than 58% of the average running time of MDA for the same set of instances. As shown in Table 2, the ratio between the average running time of DCA and that of MDA does not seem to change much as n grows. This indicates that the running time of both algorithms grows at a similar rate with the instance size. This observation does not quite match the time complexity difference between the two algorithms, which suggests the running time of DCA could grow much faster with the problem size n than MDA. As we discussed in the introduction, we hypothesize that this discrepancy is mainly caused by the worst-case analysis of the complexity of DCA. As shown by the proof of Proposition 4, the quadratic growth of running time only happens when we have to fix $n-1$ nested bound constraints at equality. Then DCA will have n recursions and the number of DRAP subproblems will grow linearly with n. *But the number of recursions needed for some instance may be significantly fewer than* n*.* This is indeed observed in our experiment for all test instances. As Table 2 shows, the average number of DRAP subproblems solved by DCA grows much more slowly than n.

For the largest $n = 6,553,600$, the number of DRAPs solved by DCA is almost negligible compared to that by MDA.

Finally, we observe that the number of DRAPs solved by MDA are the same for instances of the same size. This is due to the design of MDA, which recursively divides the problem into two subproblems with even size until the subproblem contains only one variable. This indicates that MDA solves many DRAP subproblems with a few number of variables. This also explains that despite the much larger number of DRAPs solved by MDA, its overall solution time is comparable to that of DCA.

6.3.2. DRAP-NC with objectives [CRASH] [CRASH] is a convex cost function that appears in the context of project crashing (Foldes and Soumis 1993). In each instance, the function $f_i(x)$ has the form $f_i(x) = k_i + \frac{p_i}{x}$, where p_i and k_i are drawn independently from a uniform distribution over $[0,1]$. We set $n = 2^i \times 100$ for $i \in [3:16]$ and $V_b = 100$. Other parameters are generated in the same way as in the instances with objectives [F]. We generate 10 instances for each value of n, and 140 instances in total. The average running time as well as the average number of DRAP subproblems solved are reported in Table 3 and Figure 3.

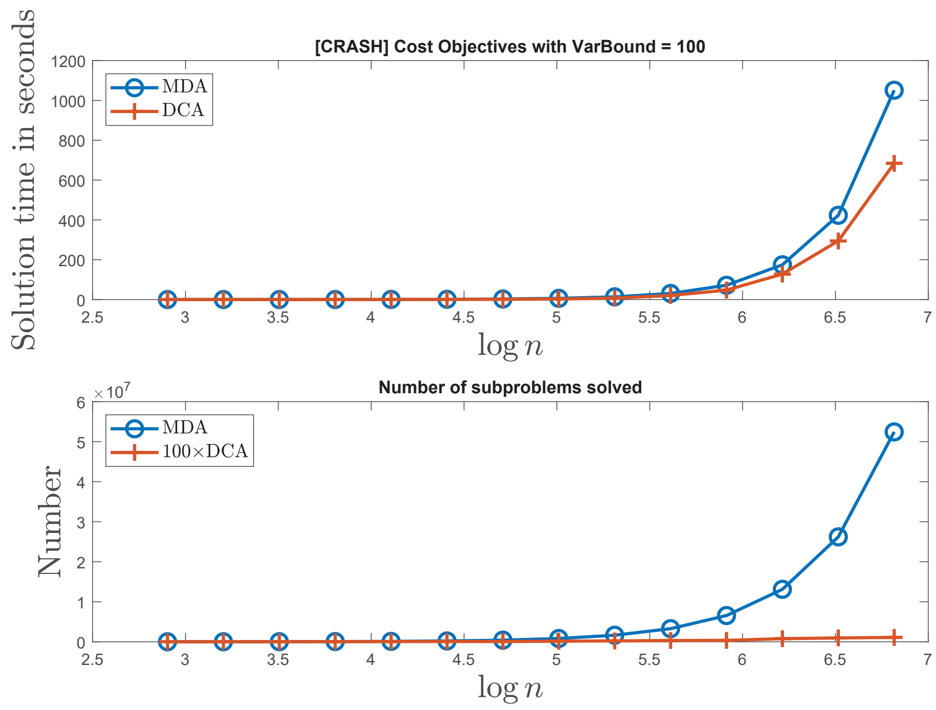

Figure 3 The growth of running time and the number of DRAP subproblems solved for instances with objectives [CRASH].

Parameters	Time (s)			Average number of DRAPs solved		
n	DCA	MDA	Ratio	DCA	MDA	Ratio
800	0.01	0.03	33%	113.0	6,396	1.77%
1,600	0.02	0.05	40%	125.4	12,796	0.98%
3,200	0.04	0.10	40%	230.4	25,596	0.90%
6,400	0.10	0.23	43%	280.0	51,196	0.55%
12,800	0.24	0.50	48%	565.0	102,396	0.55%
25,600	0.55	1.17	47%	749.8	204,796	0.37%
51,200	1.12	2.72	41%	631.2	409,596	0.15%
102,400	2.83	6.02	47%	1447.4	819,196	0.18%
204,800	6.97	13.57	51%	2224.0	1,638,396	0.14%
409,600	20.39	30.65	67%	3221.4	3,276,796	0.10%
819,200	47.39	71.72	66%	3471.2	6,553,596	0.05%
1,638,400	127.78	174.57	73%	7757.8	13,107,196	0.06%
3,276,800	294.26	423.09	70%	9618.2	26,214,396	0.04%
6,553,600	684.16	1051.07	65%	10924.6	52,428,796	0.02%

Table 3 Solution statistics of DCA and MDA for instances with objectives [CRASH].

The performance of DCA and MDA for the instances with objectives [CRASH] is similar to their performance for the instances with objectives [F]. The running time of both algorithms grows at a similar pace with the instance size, with the average running time of DCA no more than 73% of that of MDA for the same set of instances. The average number of DRAP instances solved by DCA grows much slower with n, which again implies that the worst-case running time is a very conservative estimate for this set of instances.

6.3.3. DRAP-NC with objectives [FUEL] [FUEL] is a convex function used to measure fuel consumption in vessel speed optimization (Ronen 1982, He et al. 2017). In each instance, the function $f_i(x)$ has the form $f_i(x) = p_i \times c_i \times (\frac{c_i}{x})^3$, where p_i and c_i are drawn independently from a uniform distribution over $[0,1]$. We set $n = 2^i \times 100$ for $i \in [3:16]$ and $V_b = 100$. Other parameters are generated in the same way as in the instances with objectives [F]. We generate 10 instances for each value of n, and 140 instances in total. The average running time as well as the average number of DRAP subproblems solved are reported in Table 4 and Figure 4.

Parameters	Time (s)			Average number of DRAPs solved		
n	DCA	MDA	Ratio	DCA	MDA	Ratio
800	0.01	0.03	33%	136.2	6,396	2.13%
1,600	0.02	0.05	40%	176.0	12,796	1.38%
3,200	0.05	0.11	45%	246.0	25,596	0.96%
6,400	0.11	0.25	44%	286.6	51,196	0.56%
12,800	0.26	0.57	46%	490.8	102,396	0.48%
25,600	0.58	1.29	45%	674.6	204,796	0.33%
51,200	1.44	2.80	51%	1387.8	409,596	0.34%
102,400	3.34	6.50	51%	1805.8	819,196	0.22%
204,800	8.76	14.97	59%	1950.4	1,638,396	0.12%
409,600	23.83	33.54	71%	3119.0	3,276,796	0.10%
819,200	54.56	80.23	68%	3810.2	6,553,596	0.06%
1,638,400	135.96	187.81	72%	7338.4	13,107,196	0.06%
3,276,800	303.03	445.49	68%	9568.6	26,214,396	0.04%
6,553,600	698.21	1060.27	66%	14551.8	52,428,796	0.03%

Table 4 Solution statistics of DCA and MDA for instances with objectives [FUEL].

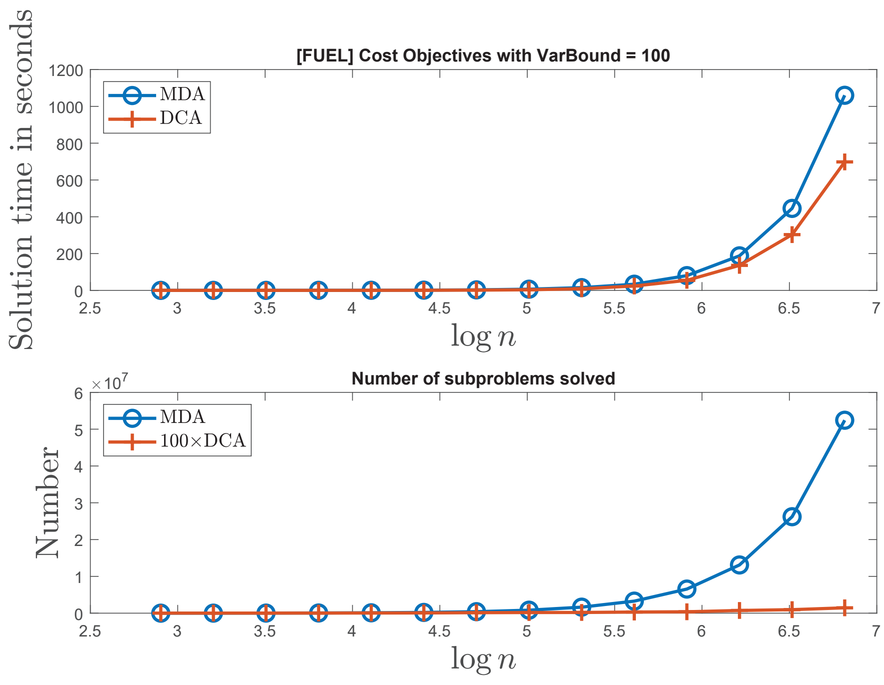

Figure 4 The growth of running time and the number of DRAP subproblems solved for instances with objectives [FUEL].

The performance of DCA and MDA for the instances with objectives [FUEL] is again very similar to their performance for the instances with objectives [F] and [CRASH]. The running time of both algorithms grows at a similar pace with the instance size, with the average running time of DCA no more than 72% of that of MDA for the same set of instances. As the instance size grows, the average number of DRAP subproblems solved by DCA is negligible compared to that by MDA.

From the experiment above, we observe that the performance of DCA is very similar in the three classes of instances: the average running time grows slower than n^2 and the average number of DRAP subproblems solved grows slowly with n. In addition, the performance of DCA does not seem to be affected by the form of the convex costs. This is not surprising since DCA does not use any information of the function form.

7. Conclusions and future work

In the paper, we proposed a simple and efficient exact algorithm to solve the discrete resource allocation problem with nested bound constraints. This algorithm first solves the problem by relaxing the nested bound constraints, and then recursively divides the problem into two smaller subproblems of the same type by fixing a mostly violated bound constraint at equality. This simple technique turns out to be very effective in solving large-sized instances. The worst-case time complexity of our algorithm is $\Theta(\max\{n^2 \log \frac{B}{n}, n^2\})$, worse than that of the current best algorithm MDA, but the running time of our algorithm is no more than 73% of the running time of MDA on all the instances we tested. An important direction to explore is whether this infeasibility guided divide-and-conquer framework could be extended to resource allocation problems with other types of constraints, such as the generalized upper-bound constraints, tree constraints, or more general submodular constraints studied in Hochbaum (1994). Another direction worth exploring is how DCA could assist the design of exact algorithms for more complex problems such as the joint vehicle routing and speed optimization problem (Fukasawa et al. 2018). The current most efficient exact algorithms for these types of problems usually employ a branch-and-price framework and the biggest challenge is how to design an efficient algorithm for the pricing problem. It would be interesting to see how DCA could help solve the pricing problem.

Acknowledgments

We are extremely grateful to Thibaut Vidal for proposing an initial version of Instance (6) and sharing the implementation details of MDA. His comments on an earlier version of our algorithm helped improve the quality of our manuscript greatly. We also thank the editorial team and anonymous referees for many insightful comments and suggestions. This work was done while the second author was at the Sun Yat-sen University.

References

Ahuja RK, Hochbaum DS (2008) Solving linear cost dynamic lot-sizing problems in $O(n \log n)$ time. *Operations Research* 56(1):255–261.

Bektaş T, Laporte G (2011) The pollution-routing problem. *Transportation Research Part B: Methodological* 45(8):1232–1250.

Brucker P (1984) An $O(n)$ algorithm for quadratic knapsack problems. *Operations Research Letters* 3(3):163–166.

Chu W, Keerthi SS (2007) Support vector ordinal regression. *Neural Computation* 19(3):792–815.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) *Introduction to Algorithms* (The MIT Press), 3rd edition.

Dyer M, Kayal N, Walker J (1984) A branch and bound algorithm for solving the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics* 11(2):231–249.

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

Federgruen A, Groenevelt H (1986b) Optimal flows in networks with multiple sources and sinks, with applications to oil and gas lease investment programs. *Operations Research* 34(2):218–225.

Foldes S, Soumis F (1993) PERT and crashing revisited: Mathematical generalizations. *European Journal of Operational Research* 64(2):286–294.

Frederickson GN, Johnson DB (1982) The complexity of selection and ranking in $X+Y$ and matrices with sorted columns. *Journal of Computer and System Sciences* 24(2):197–208.

Fukasawa R, He Q, Santos F, Song Y (2018) A joint vehicle routing and speed optimization problem. *INFORMS Journal on Computing* 30(4):694–709.

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

Groenevelt H (1991) Two algorithms for maximizing a separable concave function over a polymatroid feasible region. *European Journal of Operational Research* 54(2):227–236.

Gross O (1956) *A Class of Discrete Type Minimization Problems* (RM-1644, Rand Corporation).

Gurobi Optimization (2019) The Gurobi optimizer. `http://www.gurobi.com`.

He Q, Zhang X, Nip K (2017) Speed optimization over a path with heterogeneous arc costs. *Transportation Research Part B: Methodological* 104:198–214.

Hijazi H, Bonami P, Ouorou A (2014) An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS Journal on Computing* 26(1):31–44.

Hochbaum DS (1994) Lower and upper bounds for the allocation problem and other nonlinear optimization problems. *Mathematics of Operations Research* 19(2):390–409.

Ibaraki T, Katoh N (1988) *Resource Allocation Problems: Algorithmic Approaches* (MIT press).

Katoh N, Shioura A, Ibaraki T (2013) Resource allocation problems. *Handbook of Combinatorial Optimization* 2897–2988.

Koopman BO (1953) The optimum distribution of effort. *Journal of the Operations Research Society of America* 1(2):52–63.

Kramer R, Subramanian A, Vidal T, Lucídio dos Anjos FC (2015) A matheuristic approach for the pollution-routing problem. *European Journal of Operational Research* 243(2):523–539.

Kronqvist J, Bernal DE, Lundell A, Grossmann IE (2019) A review and comparison of solvers for convex minlp. *Optimization and Engineering* 20:397–455.

Kronqvist J, Lundell A, Westerlund T (2018) Reformulations for utilizing separability when solving convex minlp problems. *Journal of Global Optimization* 71(3):571–592.

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(4):558–625.

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

Patriksson M, Strömberg C (2015) Algorithms for the continuous nonlinear resource allocation problem—new implementations and numerical studies. *European Journal of Operational Research* 243(3):703–722.

Rockafellar RT (1970) *Convex Analysis* (Princeton University Press).

Ronen D (1982) The effect of oil price on the optimal speed of ships. *Journal of the Operational Research Society* 33(11):1035–1040.

van der Klauw T, Gerards ME, Hurink JL (2017) Resource allocation problems in decentralized energy management. *OR Spectrum* 39(3):749–773.

Vidal T (2018) Personal communication.

Vidal T, Gribel D, Jaillet P (2018) Separable convex optimization with nested lower and upper constraints. *INFORMS Journal on Optimization* 1(1):71–90.

Vidal T, Jaillet P, Maculan N (2016) A decomposition algorithm for nested resource allocation problems. *SIAM Journal on Optimization* 26(2):1322–1340.