Computers & Operations Research 135 (2021) 105451

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

A fast algorithm for quadratic resource allocation problems with nested constraints

Martijn H.H. Schoot Uiterkamp *, Johann L. Hurink, Marco E.T. Gerards

Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

ARTICLE INFO

Keywords:
Resource allocation
Nested constraints
Quadratic programming
Energy
Battery
Fast algorithm

ABSTRACT

We study the separable convex quadratic resource allocation problem with lower and upper constraints on nested sums of variables. This problem occurs in many applications, in particular battery scheduling within decentralized energy management (DEM) for smart grids. We present an algorithm for this problem that runs in $O(n \log n)$ time and, in contrast to existing algorithms for this problem, achieves this time complexity using relatively simple and easy-to-implement subroutines and data structures. This makes our algorithm very attractive for real-life adaptation and implementation. Numerical comparisons of our algorithm with a subroutine for battery scheduling within an existing tool for DEM research indicates that our algorithm significantly reduces the overall execution time of the DEM system, especially when the battery is expected to be completely full or empty multiple times in the optimal schedule. Moreover, computational experiments with synthetic data show that our algorithm outperforms the currently most efficient algorithm by more than one order of magnitude. In particular, our algorithm is able to solves all considered instances with up to ten million variables in less than four minutes on a personal computer.

1. Introduction

1.1. Resource allocation problems and energy management

The resource allocation problem is a classical and well-researched problem in the optimization and operations research literature. The objective of the resource allocation problem is to divide a fixed amount of resource (e.g., time, money, energy) over a set of activities while minimizing a given cost function (or maximizing a given utility function). In the most studied version of this problem, the cost functions are quadratic, which leads to the following formulation of the so-called quadratic resource allocation problem (QRAP):

$$\text{QRAP:} \quad \min_{\boldsymbol{x}\in\mathbb{R}^n} \sum_{i\in\mathcal{N}} \frac{1}{2}\frac{x_i^2}{a_i} \quad \text{s.t.} \sum_{i\in\mathcal{N}} x_i = R, \quad l_i \le x_i \le u_i, \quad i \in \mathcal{N}, \tag{1}$$

where $\boldsymbol{a} \in \mathbb{R}^n_{>0}$, $R \in \mathbb{R}$, $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$, and $\mathcal{N} := \{1, \dots, n\}$ (throughout this article, we use bold font for vectors). The problem QRAP has been studied extensively over the last decades due to its wide applicability in, among others, engineering, finance, and machine learning (see also the surveys in Patriksson (2008) and Patriksson and Strömberg (2015)). As a consequence, many efficient algorithms have been developed for this problem and its generalizations.

In this article, we study an extension of QRAP, namely the QRAP with lower and upper constraints on nested sums of variables (QRAP-NC). This problem can be formulated as follows:

$$\text{QRAP-NC:} \quad \min_{\boldsymbol{x}\in\mathbb{R}^n} \sum_{i\in\mathcal{N}} \frac{1}{2}\frac{x_i^2}{a_i} \quad \text{s.t.} \sum_{i\in\mathcal{N}} x_i = R, \quad L^j \le \sum_{i\in\mathcal{N}^j} x_i \le U^j, \quad j \in \mathcal{N}^{n-1}, \quad l_i \le x_i \le u_i, \quad i \in \mathcal{N}, \tag{2}$$

where $\mathcal{N}^j := \{1, \dots, j\}$ for $j \in \mathcal{N}\backslash\{n\}$, $\boldsymbol{L}, \boldsymbol{U} \in \mathbb{R}^{n-1}$, and we define $L^n = U^n = R$ for convenience. Note that if $L^j = U^j$ for some $j \in \mathcal{N}^{n-1}$, we may split up the problem QRAP-NC into two smaller instances of QRAP-NC that involve the variables $x_1, \dots, x_j$ and $x_{j+1}, \dots, x_n$ respectively. Thus, we assume without loss of generality that $L^j < U^j$ for all

* Correspondence to: Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

E-mail addresses: m.h.h.schootuiterkamp@utwente.nl (M.H.H. Schoot Uiterkamp), j.l.hurink@utwente.nl (J.L. Hurink), m.e.t.gerards@utwente.nl (M.E.T. Gerards).

https://doi.org/10.1016/j.cor.2021.105451
Received 15 September 2020; Received in revised form 25 June 2021; Accepted 26 June 2021
Available online 30 June 2021

$j \in \mathcal{N}^{n-1}$. Moreover, we may assume that $L^1 = l_1$, $U^1 = u_1$, and $L^j \geq L^{j-1} + l_j$ and $U^j \leq U^{j-1} + u_j$ for $j \in \mathcal{N}^{n-1}\backslash\{1\}$.

The problem QRAP-NC has numerous applications in, among others, machine learning, telecommunications, and speed optimization problems (see also the overviews in Akhil and Sundaresan (2018) and Vidal et al. (2019)). Our particular motivation for studying QRAP-NC is its application in decentralized energy management (DEM) for smart distribution grids. In DEM, the goal is to optimize the joint energy consumption of multiple devices within, e.g., a neighborhood. In a DEM system, devices optimize their own consumption locally but this local optimization is coordinated to obtain certain global objectives (hence the term "decentralized"). In the context of DEM, we are interested in optimization of storage devices such as electrical batters and heat buffers. Energy storage devices plays an important role in DEM systems since they are quite flexible in their energy usage and are thus suitable to compensate for peak consumption or production of energy in the distribution grid (see, e.g., Roberts and Sandberg (2011), Lund et al. (2016) and Zame et al. (2018)).

One important example of a device-level optimization problem within DEM is the scheduling of a battery within a neighborhood. We consider the situation where the charging and discharging of the battery has to be scheduled over a set $\mathcal{N}$ of equidistant time intervals, each of length Δt. Given the power profile $\boldsymbol{p} := (p_i)_{i\in\mathcal{N}}$ of the neighborhood, the goal is to determine for each time interval $i \in \mathcal{N}$ the charging power x_i of the battery during this interval so that the combined battery and neighborhood profile is flattened as much as possible. Aiming for this goal reduces the stress put on the grid and the risk of blackouts. The (physical) restrictions of the battery are given by a minimum and maximum charging rate $X_{\min}$ and $X_{\max}$ and a capacity D. Given the amount of energy present in the battery (the state-of-charge (SoC)) at the start and end of the scheduling horizon, denoted by S_{start} and S_{end}, we can formulate the resulting device-level optimization problem as follows (see also van der Klauw et al. (2017)):

$$
\begin{aligned}
\text{BATTERY:} \quad \min_{\boldsymbol{x}\in\mathbb{R}^n} \ & \sum_{i\in\mathcal{N}} (p_i + x_i)^2 \\
\text{s.t. } & 0 \leq S_{\text{start}} + \Delta t \sum_{i\in\mathcal{N}^j} x_i \leq D, \ j \in \mathcal{N}^{n-1}, \\
& S_{\text{start}} + \Delta t \sum_{i\in\mathcal{N}} x_i = S_{\text{end}}, \\
& X_{\min} \leq x_i \leq X_{\max}, \quad i \in \mathcal{N}.
\end{aligned}
$$

Note that this is an instance of QRAP-NC by applying the variable transform $\boldsymbol{y} := \boldsymbol{p} + \boldsymbol{x}$.

Another important example is the scheduling of a combined heat pump and buffer system. Here, a production schedule for the heat pump must be determined while for each time interval a prescribed heat demand is satisfied and the heat production profile is flattened as much as possible. Considering the same setting as for the scheduling of a neighborhood as described in the previous paragraph, this problem can be modeled as an instance of BATTERY where $\boldsymbol{y} := \boldsymbol{p} + \boldsymbol{x}$ is interpreted as the vector of heat production and $\boldsymbol{p}$ is the vector of heat demands.

An important feature within the DEM paradigm is that device-level problems have to be solved locally. This means that the corresponding device-level optimization algorithms are executed on embedded systems with limited computational power (see, e.g., Beaudin and Zareipour (2015)) that are located within, e.g., households. Since these algorithms are called multiple times with the DEM system as a subroutine, it is important that these algorithms are very efficient. Therefore, efficient and tailored device-level optimization algorithms are crucial ingredients for the real-life implementation of DEM systems. In particular, for the optimization of storage devices, this means that fast and tailored algorithms to solve QRAP-NC are crucial. For more background on DEM, we refer to Siano (2014) and Esther and Kumar (2016).

1.2. Background and contribution

An overview of several existing efficient algorithms for QRAP-NC and some of its generalizations is given in Table 1. In particular, there is a rich literature on solution approaches for QRAP-NC with only *upper* nested constraints on sums of variables, i.e., only nested constraints of the form $\sum_{i\in\mathcal{N}^j} x_i \leq U^j$, $j \in \mathcal{N}^{n-1}$ are given. This case has been studied mainly in the context of convex optimization over submodular constraints (see, e.g., Hochbaum (1994), Hochbaum and Hong (1995) and Vidal et al. (2016)). However, the literature on the general case of QRAP-NC is limited. van der Klauw et al. (2017) propose an infeasibility-guided divide-and-conquer algorithm, to which we shall refer in this article as ALG_{inf}. This algorithm solves a relaxation of the problem where the nested constraints are ignored and, subsequently, splits up the problem into two smaller instances of QRAP-NC at the variable for which the lower or upper nested constraint is violated most in the solution to the relaxation. The worst-case time complexity of this algorithm is $O(n^2)$. Furthermore, Vidal et al. (2019) propose a decomposition-based algorithm, hereafter referred to as ALG_{dec}, that solves QRAP-NC in $O(n \log n)$ time. This algorithm decomposes QRAP-NC into a hierarchy of QRAP subproblems whose single-variable bounds are optimal solutions to QRAP subproblems further down in the hierarchy. Currently, this is the most efficient algorithm for QRAP-NC.

As mentioned before, we are interested in algorithms for QRAP-NC that are fast in practice. Although the decomposition-based algorithm ALG_{dec} has a good worst-case time complexity, we observe several disadvantages of this approach that may make it less favorable in practice than its worst-case time complexity suggests:

1. Each level of recursion within ALG_{dec} solves a series of instances of QRAP whose parameters are determined by optimal solutions to multiple instances of QRAP on earlier levels. Since each instance is solved from scratch, much time is spent on initializing the subproblems.
2. ALG_{dec} achieves for each level of recursion an $O(n)$ time complexity by solving the QRAP subproblems using an $O(n)$ time algorithm such as the ones in Kiwiel (2008). These $O(n)$ time algorithms repeatedly call linear-time algorithms such as Blum et al. (1973) to find the median of a set. However, these median-find algorithms are relatively slow in practice due to a big constant factor in their complexity (Blum et al., 1973). Moreover, they are significantly more difficult to implement than simple sorting or sampling-based strategies (Kiwiel, 2005; Alexandrescu, 2017).

To alleviate these issues, we propose in this article a new algorithm for QRAP-NC, called ALG_{seq}, which has the same time complexity as ALG_{dec}, namely $O(n \log n)$, but in contrast requires only relatively simple and fast subroutines to attain this complexity. As a consequence, this algorithm is both faster in practice and easier to implement than ALG_{dec}. These are generally more important criteria for the actual adaptation of a given algorithm than the polynomial worst-case time complexity (Müller-Hannemann and Schirra, 2010). Our algorithm builds upon the monotonicity results for QRAP-NC derived in Vidal et al. (2019) and solves a sequence of QRAP subproblems that have a sequential nested structure rather than the divide-and-conquer structure of both ALG_{dec} and ALG_{inf}. More precisely, for each $j \in \mathcal{N}$, the jth subproblem involves only the first j variables $x_1, \ldots, x_j$. As a consequence, our approach can solve its first j subproblems without any knowledge on the parameters involving indices higher than j, whereas both ALG_{inf} and ALG_{dec} require all problem parameters to be known a priori. This makes our algorithm particularly useful in situations where problem parameters arrive over time. This is, e.g., the case when each variable denotes a decision for a specific time slot and all parameters related to this time slot become available only during or at the start of this time slot. Moreover, due to the nested structure, each input and bookkeeping parameter is accessed within a relatively

Table 1
Overview of several algorithms for QRAP-NC. Technically, the complexity of the algorithms of Vidal et al. (2016, 2019) is $O(n \log m)$ where m is the number of nested constraints. However, since $m = n - 1$ in QRAP-NC, this complexity resolves to $O(n \log n)$.

Article	Type	Complexity for QRAP-NC with only upper nested constraints	Complexity for general QRAP-NC
Hochbaum and Hong (1995)	Iterative multiplier search	$O(n \log n)$	n.a.
Vidal et al. (2016)	Decomposition-based	$O(n \log n)$	n.a.
van der Klauw et al. (2017) (ALG_{inf})	Relaxation-based	$O(n^2)$	$O(n^2)$
Vidal et al. (2019) (ALG_{dec})	Decomposition-based	$O(n \log n)$	$O(n \log n)$
This article (ALG_{seq})	Sequential nested decomposition-based	$O(n \log n)$	$O(n \log n)$

small time period instead of frequently throughout the entire course of the algorithm. This is beneficial for caching since this increases the number of times a value can be accessed quickly from a cache instead of relatively slowly from the main memory.

We attain the $O(n \log n)$ complexity using an efficient implementation of double-ended priority queues (Knuth, 1998; Brass, 2008) for several bookkeeping parameters. This data type supports insertion of arbitrary elements and finding and deletion of minimum and maximum elements in at most $O(\log n)$ time. Our approach requires $O(n)$ of such operations, which leads to an overall time complexity of $O(n \log n)$. Double-ended priority queues can be implemented using specialized data structures such as min–max heaps (Atkinson et al., 1986) or by a simple coupling of a standard min-heap and max-heap (see also Brass (2008)). The latter heaps are one of the most basic data structures and many efficient implementation exist for different programming languages (Brodal, 2013). Thus, we can achieve the time complexity of $O(n \log n)$ using relatively simple data structures, as opposed to ALG_{dec}, where a more involved implementation of a linear-time median algorithm is required.

Our algorithm for QRAP-NC also leads to efficient and fast algorithms for instances of QRAP-NC where we replace each term $\frac{1}{2}\frac{x_i^2}{a_i}$ by $a_i f(\frac{x_i}{a_i})$ for each $i \in \mathcal{N}$ with a given convex function f. Such a structure is present in many applications considered in the literature, in particular in most of the applications surveyed or evaluated in Akhil and Sundaresan (2018) and Vidal et al. (2019). We obtain such efficient algorithms by a reduction result in Schoot Uiterkamp et al. (2021), which states that any optimal solution to an instance of QRAP-NC is also optimal for this instance when we take as objective function $\sum_{i \in \mathcal{N}} a_i f(\frac{x_i}{a_i})$. Note that, alternatively, a slightly weaker reduction result follows from combining a result from Nagano and Aihara (2012), namely that the reduction result holds for *differentiable* convex functions when the feasible set is given by submodular constraints, with the observation that QRAP-NC is a special case of separable convex optimization over submodular constraints (see, e.g., Schoot Uiterkamp et al. (2021) and Wu et al. (2021)). As a consequence of these reduction results, our algorithm solves also such problems in $O(n \log n)$ time. This leads to faster algorithms for a wide range of practical problems, including the vessel speed optimization problem (Norstad et al., 2011; Hvattum et al., 2013) and processor scheduling with agreeable deadlines (Huang and Wang, 2009; Gerards, 2014).

We evaluate the performance of our algorithm ALG_{seq} and compare it to the state-of-the-art algorithms ALG_{inf} and ALG_{dec}. For this evaluation, we use both synthetic instances and instances of the battery scheduling problem BATTERY using real power consumption data as input. With regard to the instances of BATTERY, we compare our approach to a tailored implementation of ALG_{inf} within DEMKit, an existing simulation tool for DEM research (Hoogsteen et al., 2019). Within DEMKit, the battery scheduling problem is used as a subroutine within a distributed optimization framework that coordinates the energy consumption of multiple devices (Gerards et al., 2015). Our results indicate that the number of tight nested constraints in an optimal solution greatly influences which algorithm is faster for a given problem instance. In particular, ALG_{seq} is on average faster than ALG_{inf}, except when the percentage of tight nested constraints is relatively low (less than 2%). Moreover, the execution time of ALG_{seq} is more stable than that of ALG_{inf}, which makes our algorithm more suitable for use in DEM systems that employ a high level of parallelism (see, e.g., Hoogsteen et al. (2018)). With regard to the synthetic instances, we study the scalability of ALG_{seq}, ALG_{inf}, and ALG_{dec}. Our results indicate that both our algorithm ALG_{seq} and ALG_{inf} are at least one order of magnitude faster than ALG_{dec} and that ALG_{seq} is on average more than twice as fast as ALG_{inf}. In particular, ALG_{seq} solves instances with up to ten million variables in less than four minutes.

Summarizing, the contribution of this article is as follows:

- We present a fast and efficient algorithm for QRAP-NC that matches the best-known time complexity of $O(n \log n)$ and can attain this complexity using relatively simple and fast subroutines;
- We show that our algorithm is suitable for use in DEM systems due to its fast and stable execution time;
- We show that our algorithm outperforms the currently most efficient algorithm by at least one order of magnitude on the studied problem instances.

The outline of the remainder of this article is as follows. In Section 2, we present a simple procedure to solve QRAP, which forms an important ingredient for our eventual approach for solving QRAP-NC. In Section 3, we present an initial sequential algorithm for solving QRAP-NC with an $O(n^2)$ worst-case time complexity. Based on this algorithm, we derive in Section 4 our $O(n \log n)$ time algorithm ALG_{seq} for this problem. In Section 5, we evaluate the performance of this algorithm and compare it to the state-of-the-art algorithms. Finally, we provide our conclusions in Section 6.

2. A breakpoint search algorithm for QRAP

In this section, we discuss a simple approach to solve QRAP that belongs to the class of so-called *breakpoint search* methods (Kiwiel, 2008; Patriksson and Strömberg, 2015) that structurally search for the optimal Lagrange multiplier corresponding to the resource constraint (1). This approach forms an important ingredient of our $O(n \log n)$ time algorithm for QRAP-NC in Section 4.

We start by considering the Lagrangian relaxation of QRAP:

$$\text{QRAP}[\delta]: \min_{\boldsymbol{x} \in \mathbb{R}^n} \sum_{i \in \mathcal{N}} \left(\frac{1}{2}\frac{x_i^2}{a_i} - \delta x_i \right) + \delta R \quad \text{s.t. } l_i \le x_i \le u_i, \quad i \in \mathcal{N},$$

where $\delta \in \mathbb{R}$ is the Lagrange multiplier corresponding to the resource constraint (1). We denote the optimal solution to this problem by $\boldsymbol{x}[\delta] := (x_i[\delta])_{i \in \mathcal{N}}$. Since the objective function of this problem is separable, the optimal solution to QRAP[δ] is given by

$$x_i[\delta] = \begin{cases} l_i & \text{if } \delta < \frac{l_i}{a_i}, \\ a_i \delta & \text{if } \frac{l_i}{a_i} \le \delta < \frac{u_i}{a_i}, \\ u_i & \text{if } \frac{u_i}{a_i} \le \delta. \end{cases} \tag{3}$$

Observe that $x_i[\delta]$ is a continuous piecewise linear non-decreasing function of δ. More precisely, $x_i[\delta]$ is constant for $\delta \le \frac{l_i}{a_i}$, linear with

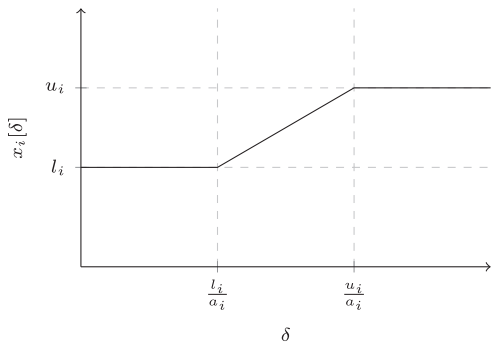


Fig. 1. The function $x_i[\delta]$ for a given $i \in \mathcal{N}$. The slope of the line segment for $\delta \in [\frac{l_i}{a_i}, \frac{u_i}{a_i}]$ is a_i.

Table 2
Updating the bookkeeping sums $P(\delta)$ and $Q(\delta)$ when searching the breakpoints in non-decreasing order.

Type of δ	Update $P(\delta)$	Update $Q(\delta)$
$\delta \equiv \alpha_i$	$P(\delta) - l_i$	$Q(\delta) + a_i$
$\delta \equiv \beta_i$	$P(\delta) + u_i$	$Q(\delta) - a_i$

slope a_i for $\delta \in [\frac{l_i}{a_i}, \frac{u_i}{a_i}]$, and again constant for $\delta \geq \frac{u_i}{a_i}$ (see also Fig. 1). For each $i \in \mathcal{N}$, we call the points where $x_i[\delta]$ has "kinks", i.e., where $x_i[\delta]$ is non-differentiable, the *breakpoints* of $x_i[\delta]$. We denote these breakpoints for $i \in \mathcal{N}$ by α_i and β_i respectively, i.e., $\alpha_i := \frac{l_i}{a_i}$ and $\beta_i := \frac{u_i}{a_i}$, where we refer to α_i as the *lower* breakpoint of $x_i[\delta]$ and to β_i as the *upper* breakpoint of $x_i[\delta]$. We denote the multiset of lower breakpoints by $\mathcal{A} := \{\alpha_i \mid i \in \mathcal{N}\}$ and the multiset of upper breakpoints by $\mathcal{B} := \{\beta_i \mid i \in \mathcal{N}\}$. The reason for defining $\mathcal{A}$ and $\mathcal{B}$ as *multi*sets is so that we can readily associate each breakpoint value in the set with one index in $\mathcal{N}$.

Note that also the sum $z[\delta] := \sum_{i \in \mathcal{N}} x_i[\delta]$ is continuous, piecewise linear, and non-decreasing. Moreover, it has $2n$ breakpoints, namely those of all terms $x_i[\delta]$. Thus, the multiset of breakpoints of $z[\delta]$ is given by $\mathcal{A} \cup \mathcal{B}$. Feasibility of the original problem QRAP implies that there exists a value $\bar{\delta}$ for the Lagrange multiplier δ such that $z[\bar{\delta}] = R$, meaning that $x[\bar{\delta}]$ is optimal not only for QRAP($\bar{\delta}$) but also for the original problem QRAP. Note that this multiplier is not necessarily unique: in general, there may exist an interval $I \subset \mathbb{R}$ such that $\delta \in I$ implies $z[\delta] = R$.

Our approach to find the value $\bar{\delta}$ consists of two steps. First, we aim to find two consecutive breakpoints δ_1 and δ_2 such that $\delta_1 \leq \bar{\delta} < \delta_2$. Since z is non-decreasing, this is equivalent to finding two consecutive breakpoints δ_1 and δ_2 such that $z[\delta_1] \leq R < z[\delta_2]$. For this, we may consider all breakpoints in $\mathcal{A} \cup \mathcal{B}$ in non-decreasing order until we have found the first, i.e., smallest, breakpoint δ such that $\bar{\delta} < \delta$. In detail, for each candidate breakpoint δ, we compute $z[\delta]$ and if $z[\delta] > R$, we set $\delta_2 := \delta$ and δ_1 as the previously considered breakpoint. To compute $z[\delta]$ efficiently, we keep track of the sums

$$P(\delta) := \sum_{i:\ \delta < \frac{l_i}{a_i}} l_i + \sum_{i:\ \delta \geq \frac{u_i}{a_i}} u_i, \qquad Q(\delta) := \sum_{i:\ \frac{l_i}{a_i} \leq \delta < \frac{u_i}{a_i}} a_i$$

and update these values each time a new breakpoint has been considered (see Table 2).

In a second step, given the consecutive breakpoints δ_1 and δ_2 with $\bar{\delta} \in [\delta_1, \delta_2)$, we determine $\bar{\delta}$ and $x[\bar{\delta}]$. If $z(\delta_1) = R$, then $\bar{\delta} = \delta_1$ and we are done. Otherwise, we have for each $i \in \mathcal{N}$:

- $x_i[\bar{\delta}] = l_i$ if and only if $x_i(\delta_2) = l_i$, and
- $x_i[\bar{\delta}] = u_i$ if and only if $x_i(\delta_1) = u_i$.

To see the first equivalence, first suppose that $x_i[\bar{\delta}] = l_i$. Since $\bar{\delta}$ is no breakpoint, we have that $\bar{\delta} < \alpha_i$. As a consequence, since there is no breakpoint between $\bar{\delta}$ and δ_2, we have that $\delta_2 \leq \alpha_i$. It follows that $x_i[\delta_2] = l_i$. Second, suppose that $x_i[\delta_2] = l_i$. Since $x[\delta]$ is non-decreasing, it follows that $l_i \leq x_i[\bar{\delta}] \leq x_i[\delta_2] = l_i$ and thus that $x_i[\bar{\delta}] = l_i$. The second equivalence holds analogously. Thus, given δ_1 and δ_2, we know whether a given variable $x_i[\bar{\delta}]$ equals its lower bound l_i, its upper bound u_i, or is strictly in between these bounds. To find $x_i[\bar{\delta}]$ for those variables that are strictly in between their bounds, note that, by definition of $x[\delta]$,

$$R = z[\bar{\delta}] = \sum_{i:\ x_i[\bar{\delta}] = l_i} l_i + \sum_{i:\ l_i < x_i[\bar{\delta}] < u_i} a_i \bar{\delta} + \sum_{i:\ x_i[\bar{\delta}] = u_i} u_i.$$

It follows that

$$\bar{\delta} = \frac{R - \sum_{i:\ x_i[\bar{\delta}] = l_i} l_i - \sum_{i:\ x_i[\bar{\delta}] = u_i} u_i}{\sum_{i:\ l_i < x_i[\bar{\delta}] < u_i} a_i},$$

from which we can directly compute $x_i[\bar{\delta}]$ by $x_i[\bar{\delta}] = a_i \bar{\delta}$.

Algorithm 1 summarizes the sketched approach. To efficiently compute the minimum breakpoint δ_k, we can implement the multisets $\mathcal{A}$ and $\mathcal{B}$ as sorted lists. As a consequence, each iteration of the algorithm takes $O(1)$ time. Since the maximum number of iterations is $2n$ (one for each breakpoint), the overall complexity of this approach is $O(n \log n)$ due to the initial sorting of the breakpoints. If this sorting is given (for example if the breakpoints have already been sorted in a previous run of the algorithm), the time complexity of the algorithm reduces to $O(n)$.

Algorithm 1 An $O(n \log n)$ time algorithm for QRAP.

Input: Parameters $\boldsymbol{a} \in \mathbb{R}^n_{>0}$, $R \in \mathbb{R}$, and $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$
Output: Optimal solution $\bar{\boldsymbol{x}}$ to QRAP
Compute the breakpoint multisets $\mathcal{A}$ and $\mathcal{B}$
Initialize $P := \sum_{i \in \mathcal{N}} l_i$; $Q := 0$
repeat
  Determine smallest breakpoint $\delta_i := \min(\mathcal{A} \cup \mathcal{B})$
  if $P + Q\delta_i = R$ **then**
    $\bar{\delta} = \delta_i$; compute $x[\bar{\delta}]$ using Equation (3)
    return
  else if $P + Q\delta_i > R$ **then** $\{\bar{\delta} < \delta_i\}$
    $\bar{\delta} = \frac{R-P}{Q}$; compute $x[\bar{\delta}]$ using Equation (3)
    return
  else
    if δ_i is lower breakpoint ($\delta_i = \alpha_i$) **then**
      $P := P - l_i$; $Q := Q + a_i$
      $\mathcal{A} := \mathcal{A} \backslash \{\alpha_i\}$
    else
      $P := P + u_i$; $Q := Q - a_i$
      $\mathcal{B} := \mathcal{B} \backslash \{\beta_i\}$
    end if
  end if
until multiplier $\bar{\delta}$ has been found
return Optimal solution $\bar{\boldsymbol{x}} := \boldsymbol{x}[\bar{\delta}]$

We conclude this subsection with two observations that are crucial for the efficiency of our algorithm for QRAP-NC presented in the following section:

1. Instead of searching the breakpoints in non-decreasing order, we may also search them in *non-increasing order* and continue the search until we find the first, i.e., largest breakpoint δ_1 such that $\delta_1 < \bar{\delta}$.
2. Solving two instances of QRAP that differ only in the value of R in the resource constraint (1) can be done simultaneously in one run of Algorithm 1. This is because the multisets of the breakpoints for these two instances of QRAP are the same. Thus, we can modify Algorithm 1 such that it continues the breakpoint search after the optimal multiplier for the smallest of the given values of R has been found. Note that, essentially, the optimal multiplier for a given value R serves as the starting

candidate for the optimal multiplier for instances with a higher value of R. This is in fact one of the two crucial observations for our approach for solving QRAP-NC, which we discuss further in Section 4.2.

3. An initial sequential algorithm for QRAP-NC

In this section, we present our initial sequential algorithm for the problem QRAP-NC. This algorithm solves the problem as a sequence of $2n-1$ instances of QRAP whose single-variable bounds (2) are optimal solutions to previous QRAP subproblems. For this, we consider a sequence of restricted subproblems where we take into account only a subset of the variables. More precisely, we define for each $j \in \mathcal{N}$ and $C \in \mathbb{R}$ the following subproblem:

$$\begin{aligned}
\text{QRAP-NC}^j(C) \quad : \quad & \min_{\boldsymbol{x}\in\mathbb{R}^j} \sum_{i\in\mathcal{N}^j} \frac{1}{2}\frac{x_i^2}{a_i} \\
& \text{s.t.} \sum_{i\in\mathcal{N}^j} x_i = C, && (4)\\
& L^k \le \sum_{i\in\mathcal{N}^k} x_i \le U^k, \quad k \in \mathcal{N}^{j-1}, && (5)\\
& l_i \le x_i \le u_i, \quad i \in \mathcal{N}^j.
\end{aligned}$$

Throughout this article, we denote the optimal solution to this subproblem by $\boldsymbol{x}^j(C) := (x_i^j(C))_{i\in\mathcal{N}^j}$, where we use the brackets $(\cdot)$ instead of $[\cdot]$ to emphasize the distinction of this solution from an optimal solution $\boldsymbol{x}[\delta]$ of the Lagrangian relaxation QRAP(δ) of QRAP. Note that this optimal solution is unique since the objective function of the corresponding problem is *strictly* convex and all constraints are linear. Moreover, observe that the nth subproblem QRAP-NC$^n(R)$ is equal to the original problem QRAP-NC.

The key ingredient to our algorithm is that we can replace the nested constraints (5) by specific single-variable constraints without changing the optimal solution. By doing this, we transform an instance of QRAP-NC into an equivalent instance of QRAP. More precisely, we show that each subproblem QRAP-NC$^j(C)$ yields the same optimal solution as the following instance of QRAP:

$$\begin{aligned}
\text{QRAP}^j(C) \quad : \quad & \min_{\boldsymbol{y}\in\mathbb{R}^j} \sum_{i\in\mathcal{N}^j} \frac{1}{2}\frac{y_i^2}{a_i} \\
& \text{s.t.} \sum_{i\in\mathcal{N}^j} y_i = C, && (6)\\
& x_i^{j-1}(L^{j-1}) \le y_i \le x_i^{j-1}(U^{j-1}), \quad i \in \mathcal{N}^{j-1}, && (7)\\
& l_j \le y_j \le u_j, && (8)
\end{aligned}$$

where the bounds $\boldsymbol{x}^{j-1}(L^{j-1})$ and $\boldsymbol{x}^{j-1}(U^{j-1})$ in (7) are the optimal solutions of the problems QRAP-NC$^{j-1}(L^{j-1})$ and QRAP-NC$^{j-1}(U^{j-1})$ respectively. Note that the single-variable bounds for x_j in (8) are the same as those of the original subproblem QRAP-NC$^j(C)$.

The validity of this transformation is proven by Lemmas 1–3. First, Lemma 1 shows that the optimal solution $\boldsymbol{x}^j(C)$ to the subproblem QRAP-NC$^j(C)$ is non-decreasing in C. Subsequently, Lemma 2 uses this property to show that when adding the alternative single-variable bounds (7) to the problem formulation of QRAP-NC$^j(C)$, the optimal solution $\boldsymbol{x}^j(C)$ to QRAP-NC$^j(C)$ is not cut off. Finally, Lemma 3 shows that the alternative single-variable bounds (7) are stronger than the nested constraints (5).

Lemma 1. *If $L^j \le A \le B \le U^j$, we have $\boldsymbol{x}^j(A) \le \boldsymbol{x}^j(B)$ for a given $j \in \mathcal{N}$.*

Proof. This proof is based on the proof of Theorem 2 in Vidal et al. (2019) and given in Appendix A.1. □

Lemma 2. *For a given $j \in \mathcal{N}^{n-1}$ and $C \in [L^j, U^j]$, we have that $x_i^j(L^j) \le x_i^{j+1}(C) \le x_i^j(U^j)$.*

Proof. Let $\boldsymbol{x}' := (x_1^{j+1}(C), \dots, x_j^{j+1}(C))$ be the vector of the first j components of the optimal solution to the problem QRAP-NC$^{j+1}(C)$. Since $\boldsymbol{x}'$ is feasible for all nested constraints (5) for $k \in \mathcal{N}^j$, this vector is also the optimal solution to QRAP-NC$^j(A)$ where $A := \sum_{i\in\mathcal{N}^j} x_i'$, i.e., we have $\boldsymbol{x}' = \boldsymbol{x}^j(A)$. Since $A \in [L^j, U^j]$, Lemma 1 implies that $x_i^j(L^j) \le x_i^j(A) \le x_i^j(U^j)$ for all $i \in \mathcal{N}^j$. It follows that $x_i^j(L^j) \le x_i^{j+1}(C) \le x_i^j(U^j)$ for all $i \in \mathcal{N}^j$. □

Lemma 3. *If for a given $j \in \mathcal{N}^{n-1}$ and vector $\boldsymbol{y} \in \mathbb{R}^j$ we have $\boldsymbol{x}^j(L^j) \le \boldsymbol{y} \le \boldsymbol{x}^j(U^j)$, then $L^k \le \sum_{i\in\mathcal{N}^k} y_i \le U^k$ for all $k \in \mathcal{N}^j$.*

Proof. The sum of the inequalities $x_i^j(L^j) \le y_i \le x_i^j(U^j)$ over all $i \in \mathcal{N}^k$ yields

$$\sum_{i\in\mathcal{N}^k} x_i^j(L^j) \le \sum_{i\in\mathcal{N}^k} y_i \le \sum_{i\in\mathcal{N}^k} x_i^j(U^j).$$

Since $\boldsymbol{x}^j(L^j)$ and $\boldsymbol{x}^j(U^j)$ are feasible for QRAP-NC$^j(L^j)$ and QRAP-NC$^j(U^j)$ respectively and $k \le j$, we have $L^k \le \sum_{i\in\mathcal{N}^k} x_i^j(L^j)$ and $\sum_{i\in\mathcal{N}^k} x_i^j(U^j) \le U^k$ and the result of the lemma follows. □

Lemma 2 implies that, given optimal solutions $\boldsymbol{x}^{j-1}(L^{j-1})$ and $\boldsymbol{x}^{j-1}(U^{j-1})$, we can replace the nested constraints (5) in QRAP-NC$^j(C)$ by the single-variable bounds (7) without cutting off the optimal solution to QRAP-NC$^j(C)$. Moreover, since these single-variable bounds are stronger than the nested constraints by Lemma 3, adding these constraints does not change the optimal objective value. It follows directly that any optimal solution to QRAP$^j(C)$ is also optimal for QRAP-NC$^j(C)$.

Based on Lemmas 1–3, the following approach can be used to solve QRAP-NC. We successively solve the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ from $j = 1$ to $n-1$ and finally the subproblem QRAP$^n(R)$, whereby in each step we use the optimal solutions to the preceding subproblems QRAP$^{j-1}(L^{j-1})$ and QRAP$^{j-1}(U^{j-1})$ as input. Note that each of the subproblems is an instance of QRAP. This approach is summarized in Algorithm 2.

Algorithm 2 An initial sequential algorithm for QRAP-NC.

1: **Input:** Parameters $\boldsymbol{a} \in \mathbb{R}^n_{>0}$, $\boldsymbol{L}, \boldsymbol{U} \in \mathbb{R}^{n-1}$, $R \in \mathbb{R}$, and $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$
2: **Output:** Optimal solution $\bar{\boldsymbol{x}}$ to QRAP-NC
3: Initialize $x_1^1(L^1) = L^1$; $x_1^1(U^1) = U^1$
4: **for** $j = 2, \dots, n-1$ **do**
5: Compute optimal solutions $\boldsymbol{x}^j(L^j)$ and $\boldsymbol{x}^j(U^j)$ to QRAP$^j(L^j)$ and QRAP$^j(U^j)$ respectively
6: **end for**
7: Compute optimal solution $\boldsymbol{x}^n(R)$ to QRAP$^n(R)$
8: **return** Optimal solution $\bar{\boldsymbol{x}} := \boldsymbol{x}^n(R)$

Since each subproblem QRAP$^j(\cdot)$ can be solved in $O(n)$ time (Brucker, 1984), the worst-case time complexity of Algorithm 2 is $O(n^2)$. However, linear-time algorithms for QRAP such as Brucker (1984) attain their linear time complexity by employing linear-time algorithms for median finding, which are, as already mentioned, in general slower than simple sorting- or sampling-based approaches (Kiwiel, 2005; Alexandrescu, 2017). Note, that also the $O(n \log n)$ time algorithm ALG$_{\text{dec}}$ attains its worst-case time complexity by using such slow linear-time algorithms as a subroutine.

In the next section, we propose an algorithm to solve QRAP-NC in $O(n \log n)$ time that, as opposed to ALG$_{\text{dec}}$, does not require linear-time median-finding algorithms. Instead, it only requires a simple data structure for double-ended priority queues to store several bookkeeping parameters.

We conclude this section with two remarks that may be of independent interest:

1. It can be shown that Lemmas 1–3 also hold for the case where the variables are integer-valued, i.e., $\boldsymbol{x} \in \mathbb{Z}^n$ (see also Theorem 5 in Vidal et al. (2019)), given that all parameters $\boldsymbol{a}$, $\boldsymbol{L}$, $\boldsymbol{U}$, $\boldsymbol{l}$, and $\boldsymbol{u}$ are also integer-valued and nonnegative. As a consequence, when solving each subproblem QRAP$^j(\cdot)$ with integer variables, Algorithm 2 computes an optimal solution to QRAP-NC with integer variables. The worst-case time complexity of this algorithm is $O(n^2)$ since each QRAP$^j(\cdot)$ subproblem with integer variables can be solved in $O(j)$ time (Ibaraki and Katoh, 1988).
2. Lemmas 1–3 can be generalized to the case where the objective function is the sum of separable convex cost functions f_i, i.e., where we replace each term $\frac{1}{2}\frac{x_i^2}{a_i}$ by a convex function $f_i(x_i)$. For this more general problem, this leads to a sequential algorithm that is very similar to Algorithm 2. However, initial computational tests indicated that both this algorithm and Algorithm 2 are in practice much slower than both ALG_{inf} and ALG_{dec}.

4. A fast $O(n \log n)$ time algorithm for QRAP-NC

The sequential algorithm derived in the previous section does not match the best known time complexity of the algorithm in Vidal et al. (2019). However, we show in this section that we can implement Algorithm 2 such that its time complexity reduces to $O(n \log n)$ without requiring a linear-time median finding algorithm. Instead, we only require a data type that supports insertion of elements and the finding and removing of minimum and maximum elements in $O(\log n)$ time such as a double-ended priority queue.

The key to efficiency in our approach is that we do not explicitly compute the solution to each QRAP subproblem. Instead, we only compute an optimal Lagrange multiplier corresponding to the resource constraint (6) that characterizes the entire optimal solution to this subproblem. Subsequently, we use these multipliers to reconstruct the optimal solution to the original problem QRAP-NC using two sets of simple recursive relations that can be executed in $O(n)$ time. In order to compute the Lagrange multipliers without explicitly storing intermediate solutions, we exploit the special structure of these multipliers and of a specific algorithm for solving QRAP.

First, in Section 4.1, we introduce some of the used notation. Second, in Section 4.2, we derive an efficient approach for computing the optimal Lagrange multipliers of the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$. Based on these optimal Lagrange multipliers, we derive in Section 4.3, two simple recursions to compute the optimal solution x to QRAP-NC. Finally, in Section 4.4, we present an $O(n \log n)$ algorithm for QRAP-NC and discuss an implementation that attains this worst-case time complexity.

4.1. Notation

We introduce the following notation concerning the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ that we use throughout the remainder of this article (see also Table 3). We denote for $j \in \mathcal{N}$ the lower and upper single-variable bounds (7) and (8) of QRAP$^j(C)$ with $C \in [L^j, U^j]$ by $\bar{\boldsymbol{l}}^j := (\bar{l}_i^j)_{i \in \mathcal{N}^j}$ and $\bar{\boldsymbol{u}}^j := (\bar{u}_i^j)_{i \in \mathcal{N}^j}$, where $\bar{l}_i^j := x_i^{j-1}(L^{j-1})$ and $\bar{u}_i^j := x_i^{j-1}(U^{j-1})$ for $i < j$, and $\bar{l}_j^j := l_j$ and $\bar{u}_j^j := u_j$. Furthermore, we denote by $\boldsymbol{\alpha}^j := (\alpha_i^j)_{i \in \mathcal{N}^j}$ and $\boldsymbol{\beta}^j := (\beta_i^j)_{i \in \mathcal{N}^j}$ the lower and upper breakpoints for the QRAP$^j(C)$ subproblem. We call the breakpoints corresponding to $i = j$, i.e., α_j^j and β_j^j, *initial* breakpoints since QRAP$^j(C)$ is the first subproblem, i.e., with lowest index j, in which we have to compute breakpoint values for the variable x_j. Note that we can compute these breakpoints directly as $\alpha_j^j := \frac{l_j}{a_j}$ and $\beta_j^j := \frac{u_j}{a_j}$ by definition of the subproblem QRAP$^j(C)$.

Furthermore, let κ^j and λ^j denote the optimal Lagrange multipliers for the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ respectively and define $\boldsymbol{\kappa} := (\kappa^j)_{j \in \mathcal{N}}$ and $\boldsymbol{\lambda} := (\lambda^j)_{j \in \mathcal{N}}$, where we set $\kappa^1 := \alpha_1^1$ and $\lambda^1 := \beta_1^1$. If the optimal Lagrange multiplier for a given subproblem QRAP$^j(L^j)$ is not unique, we define without loss of generality κ^j as the *maximum* optimal Lagrange multiplier. Analogously, we define λ^j as the *minimum* optimal Lagrange multiplier of subproblem QRAP$^j(U^j)$. Note that $\boldsymbol{x}^j(L^j) = \boldsymbol{x}^j[\kappa^j]$ and $\boldsymbol{x}^j(U^j) = \boldsymbol{x}^j[\lambda^j]$ by definition of the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ and of κ^j and λ^j. Finally, for a given subproblem QRAP$^j(C)$, we define the set of its lower breakpoints as $\mathcal{A}^j := \{\alpha_i^j \mid i \in \mathcal{N}^j\}$ and the set of its upper breakpoints as $\mathcal{B}^j := \{\beta_i^j \mid i \in \mathcal{N}^j\}$. Recall that in Section 2 we defined breakpoint sets as *multi*sets for convenience when solving QRAP. However, for our approach for a fast algorithm for QRAP-NC, it is crucial that the breakpoint sets do not contain duplicate elements. Therefore, in this section and the remainder of this article, we regard $\mathcal{A}^j$ and $\mathcal{B}^j$ as ordinary sets.

Table 3
Overview of the used notation with regard to the subproblems QRAP$^j(L^j)$, QRAP$^j(U^j)$, and QRAP$^j(C)$ for $C \in [L^j, U^j]$.

Symbol	Definition
$\bar{\boldsymbol{l}}^j := (\bar{l}_i^j)_{i \in \mathcal{N}^j}$	Lower single-variable bounds of QRAP$^j(C)$
$\bar{\boldsymbol{u}}^j := (\bar{u}_i^j)_{i \in \mathcal{N}^j}$	Lower single-variable bounds of QRAP$^j(C)$
$\boldsymbol{\alpha}^j := (\alpha_i^j)_{i \in \mathcal{N}^j}$	Lower breakpoints for QRAP$^j(C)$
$\boldsymbol{\beta}^j := (\beta_i^j)_{i \in \mathcal{N}^j}$	Upper breakpoints for QRAP$^j(C)$
α_j^j, β_j^j	Initial breakpoints of QRAP$^j(C)$
κ^j	Optimal Lagrange multiplier for QRAP$^j(L^j)$
λ^j	Optimal Lagrange multiplier for QRAP$^j(U^j)$
$\mathcal{A}^j$	Set of lower breakpoints of QRAP$^j(C)$
$\mathcal{B}^j$	Set of upper breakpoints of QRAP$^j(C)$

4.2. Computing the optimal Lagrange multipliers of the subproblems

The goal of this subsection is to derive an efficient approach for computing the optimal Lagrange multiplier of each QRAP subproblem in Algorithm 2 without explicitly calculating any of the intermediate optimal solutions $\boldsymbol{x}^j(L^j)$ and $\boldsymbol{x}^j(U^j)$ for $j \in \mathcal{N}$. If we would follow the latter strategy, i.e., if we solve each pair of subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ from scratch, e.g., using Algorithm 1, we would have to explicitly compute the breakpoint sets for each pair of subproblems. This leads to $O(n^2)$ computations and thus forms an efficiency bottleneck within this algorithm.

We show that we can apply the breakpoint search procedure in Algorithm 1 for solving the subproblems such that each breakpoint set $\mathcal{A}^{j+1}$ can be obtained from the previous set $\mathcal{A}^j$ in $O(1)$ amortized steps, i.e., the total number of steps required to carry out this construction for all $j \in \mathcal{N}^{n-1}$ is $O(n)$. This can be done because of two intermediate results that we show in this subsection. First, the number of distinct values that the breakpoints can take is not $O(n^2)$ but $O(n)$. We obtain this result by unveiling a useful relation between breakpoints of consecutive subproblems, i.e., between $\boldsymbol{\alpha}^j$, $\boldsymbol{\beta}^j$ and $\boldsymbol{\alpha}^{j+1}$, $\boldsymbol{\beta}^{j+1}$. Second, when constructing the breakpoint sets, each distinct breakpoint value is included in or removed from a breakpoint set at most twice during the entire procedure. For this, it is important that we solve each lower subproblem QRAP$^j(L^j)$ by considering the breakpoints in non-decreasing order and each upper subproblem QRAP$^j(U^j)$ by considering the breakpoints in non-increasing order. Together, these two results imply that the construction of the breakpoint sets requires in total $O(n)$ additions and removals of breakpoint values. By using an appropriate data structure such as double-ended priority queues for maintaining the breakpoint sets, each of these steps can be executed in $O(\log n)$ time, which leads to an overall $O(n \log n)$ complexity for computing the optimal Lagrange multipliers $\boldsymbol{\kappa}$ and $\boldsymbol{\lambda}$.

The outline of the remainder of this subsection is as follows. First, in Section 4.2.1, we analyze the relation between breakpoints of consecutive subproblems and show that the number of distinct breakpoint values is $O(n)$. Subsequently, in Section 4.2.2, we use this information and the structure of Algorithm 1 to construct the breakpoint sets for

each subproblem from those of the preceding subproblems. Finally, in Section 4.2.3, we discuss how the updating of the bookkeeping parameters within the breakpoint search procedure must be adjusted when applying this procedure to the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$.

4.2.1. Relation between consecutive breakpoints

We first show how we can efficiently obtain the breakpoint set of a given subproblem QRAP$^{j+1}(C)$ based on the breakpoint set and optimal Lagrange multipliers of the preceding subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$. We establish for a given $j \in \mathcal{N}^{n-1}$ and $i < j$ the following relation between the subsequent lower breakpoints α_i^j and α_i^{j+1}:

- If $\kappa^j < \alpha_i^j$, it follows from Eq. (3) that $x_i^j[\kappa^j] = \bar{l}_i^j$ since $\alpha_i^j = \frac{\bar{l}_i^j}{a_i}$. This implies that $\bar{l}_i^{j+1} = x_i^j(L^j) = x_i^j[\kappa^j] = \bar{l}_i^j$ and thus $\alpha_i^{j+1} = \alpha_i^j$.
- If $\alpha_i^j \le \kappa^j < \beta_i^j$, it follows from Eq. (3) that $x_i^j[\kappa^j] = a_i\kappa^j$. Thus, $\alpha_i^{j+1} = \frac{\bar{l}_i^{j+1}}{a_i} = \frac{x_i^j(L^j)}{a_i} = \frac{x_i^j[\kappa^j]}{a_i} = \kappa^j$.
- If $\beta_i^j \le \kappa^j$, then it follows from Eq. (3) that $x_i^j[\kappa^j] = \bar{u}_i^j$. This implies that $\bar{l}_i^{j+1} = x_i^j(L^j) = x_i^j[\kappa^j] = \bar{u}_i^j$ and thus $\alpha_i^{j+1} = \beta_i^j$.

Summarizing, we can determine α_i^{j+1} from the previous breakpoints α_i^j and β_i^j and the optimal Lagrange multiplier κ^j as follows:

$$\alpha_i^{j+1} = \begin{cases} \alpha_i^j & \text{if } \kappa^j < \alpha_i^j, \\ \kappa^j & \text{if } \alpha_i^j \le \kappa^j < \beta_i^j, \\ \beta_i^j & \text{if } \beta_i^j \le \kappa^j. \end{cases} \tag{9}$$

Analogously, we obtain the following expression for the upper breakpoint β_i^{j+1} in terms of the previous breakpoints α_i^j and β_i^j and the optimal Lagrange multiplier λ^j:

$$\beta_i^{j+1} = \begin{cases} \beta_i^j & \text{if } \lambda^j > \beta_i^j, \\ \lambda^j & \text{if } \beta_i^j \ge \lambda^j > \alpha_i^j, \\ \alpha_i^j & \text{if } \alpha_i^j \ge \lambda^j. \end{cases} \tag{10}$$

Note that it follows from these relations that $\alpha_i^j \le \alpha_i^{j+1}$ and $\beta_i^j \ge \beta_i^{j+1}$ for each $j \in \mathcal{N}^{n-1}$. Moreover, note that the only values that the breakpoints α_i^{j+1} and β_i^{j+1} can take are those of the initial breakpoints α_i^i and β_i^i or of the optimal Lagrange multipliers in $\boldsymbol{\kappa}$ and $\boldsymbol{\lambda}$. More precisely, by applying Eqs. (9) and (10) recursively, we obtain the set of all possible values of α_i^{j+1} as follows:

$$\begin{aligned} \alpha_i^{j+1} &\in \{\kappa^j, \alpha_i^j, \beta_i^j\} \subseteq \{\kappa^j, \kappa^{j-1}, \alpha_i^{j-1}, \beta_i^{j-1}, \lambda^{j-1}\} \\ &\subseteq \{\kappa^j, \kappa^{j-1}, \kappa^{j-2}, \alpha_i^{j-2}, \beta_i^{j-2}, \lambda^{j-2}, \lambda^{j-1}\} \\ &\subseteq \cdots \subseteq \{\kappa^j, \ldots, \kappa^i, \alpha_i^i, \beta_i^i, \lambda^i, \ldots, \lambda^{j-1}\}. \end{aligned}$$

Analogously, we have that $\beta_i^{j+1} \in \{\kappa^{j-1}, \ldots, \kappa^i, \alpha_i^i, \beta_i^i, \lambda^i, \ldots, \lambda^j\}$. This implies that the number of distinct values among all breakpoints is limited by $4n$.

4.2.2. Constructing consecutive breakpoint sets

As observed at the end of Section 2, we can solve a given QRAP subproblem by searching its breakpoints either in non-decreasing or non-increasing order. In particular, we can solve all lower subproblems QRAP$^j(L^j)$ by searching the breakpoints in non-decreasing order and all upper subproblems QRAP$^j(U^j)$ by searching the breakpoints in non-increasing order. When doing this, note that for solving the upper subproblem QRAP$^j(U^j)$ we can use as breakpoint sets the sets that "remain" from the breakpoint search for the lower subproblem. More precisely, instead of the sets $\mathcal{A}^j$ and $\mathcal{B}^j$ that we also use as breakpoint sets for the lower subproblem QRAP$^j(L^j)$, we can use the sets $\{\alpha_i^j \in \mathcal{A}^j \mid \alpha_i^j \ge \kappa^j\}$ and $\{\beta_i^j \in \mathcal{B}^j \mid \beta_i^j \ge \kappa^j\}$ respectively. This is because $\kappa^j \le \lambda^j$ and thus in the breakpoint search for the upper problem QRAP$^j(U^j)$ no breakpoints smaller than κ^j need to be considered.

We define the sets $\tilde{\mathcal{A}}^j$ and $\tilde{\mathcal{B}}^j$ as the sets of lower and upper breakpoints that remain to be considered after solving the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ in the way described in the previous paragraph, i.e., we have

$$\begin{aligned} \tilde{\mathcal{A}}^j &:= \{\alpha_i^j \in \mathcal{A}^j \mid \kappa^j \le \alpha_i^j \le \lambda^j\}, \\ \tilde{\mathcal{B}}^j &:= \{\beta_i^j \in \mathcal{B}^j \mid \kappa^j \le \beta_i^j \le \lambda^j\}. \end{aligned}$$

We call these sets the *remaining breakpoint sets* of the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$. In the following, we relate these two remaining breakpoint sets to the breakpoint sets of the next two subproblems, i.e., to the sets $\mathcal{A}^{j+1}$ and $\mathcal{B}^{j+1}$. For this, we focus on the relation between the lower remaining breakpoint sets $\tilde{\mathcal{A}}^j$ and the lower breakpoint set $\mathcal{A}^{j+1}$; the relation between the upper remaining breakpoint set $\tilde{\mathcal{B}}^j$ and the upper breakpoint set $\mathcal{B}^{j+1}$ is analogous.

For each $i \in \mathcal{N}^j$, we determine the value of α_i^{j+1} based on the values of α_i^j, and β_i^j compared to those of κ^j, λ^j. More precisely, we consider the following four cases:

1. If $\kappa^j \le \alpha_i^j \le \lambda^j$, it follows from Eq. (9) that $\alpha_i^{j+1} = \alpha_i^j$. Thus, all values in $\tilde{\mathcal{A}}^j$ act as breakpoint values for the next subproblems, i.e., $\tilde{\mathcal{A}}^j \subseteq \mathcal{A}^{j+1}$.
2. If $\alpha_i^j < \kappa^j$ and $\kappa^j < \beta_i^j$, it follows from Eq. (9) that $\alpha_i^{j+1} = \kappa^j$.
3. If $\alpha_i^j < \kappa^j$ and $\beta_i^j \le \kappa^j$, it follows from Eqs. (9) and (10) that $x_i^j(L^j) = \bar{u}_i^j$ and $\beta_i^{j+1} = \beta_i^j = \alpha_i^{j+1}$ respectively. Thus,

   $$\beta_i^{j+1} = \alpha_i^{j+1} \le \cdots \le \alpha_i^n \le \beta_i^n \le \cdots \le \beta_i^{j+1}.$$

   This means that $\alpha_i^{j'} = \beta_i^{j'} = \beta_i^j$ and $\bar{l}_i^{j'} = \bar{u}_i^{j'} = \bar{u}_i^j$ for all $j' > j$. Thus, in all remaining subproblems, the lower and upper breakpoints of i coincide and $x_i^{j'}(C) = \bar{u}_i^j$ for any $j' > j$ and $L^{j'} \le C \le U^{j'}$, regardless of the values of the future optimal Lagrange multipliers $\kappa^{j'}$ and $\lambda^{j'}$. This means that we can remove this index (variable) from the breakpoint search.
4. Finally, if $\alpha_i^j > \lambda^j$, it follows from Eqs. (9) and (10) that $\alpha_i^{j+1} = \alpha_i^j$ and $\beta_i^{j+1} = \alpha_i^j$ respectively. Thus, $\alpha_i^{j+1} = \beta_i^{j+1} = \alpha_i^j$. Analogously to the case $\alpha_i^j \le \beta_i^j < \kappa^j$, it follows that $\bar{l}_i^{j'} = \bar{u}_i^{j'} = \bar{l}_i^j$ and $x_i^{j'}(C) = \bar{l}_i^j$ for all $j' > j$ and $L^{j'} \le C \le U^{j'}$. Thus, also in this case we can remove the index i from the breakpoint search.

Note that these four cases are mutually exclusive and cover all possible values of α_i^j and β_i^j (see also Table 4). These four cases imply that we can construct $\mathcal{A}^{j+1}$ from $\tilde{\mathcal{A}}^j$ as follows:

$$\mathcal{A}^{j+1} = \tilde{\mathcal{A}}^j \cup \{\alpha_{j+1}^{j+1}\} \cup \begin{cases} \{\kappa^j\} & \text{if there exists } i \text{ such that } \alpha_i^j < \kappa^j < \beta_i^j, \\ \emptyset & \text{otherwise.} \end{cases}$$

Analogously, we can construct $\mathcal{B}^{j+1}$ from $\tilde{\mathcal{B}}^j$ as follows:

$$\mathcal{B}^{j+1} = \tilde{\mathcal{B}}^j \cup \{\beta_{j+1}^{j+1}\} \cup \begin{cases} \{\lambda^j\} & \text{if there exists } i \text{ such that } \alpha_i^j < \lambda^j < \beta_i^j, \\ \emptyset & \text{otherwise.} \end{cases}$$

The above constructions show how the breakpoint sets evolve from j to $j+1$, i.e., how we obtain $\mathcal{A}^{j+1}$ and $\mathcal{B}^{j+1}$ from $\mathcal{A}^j$ and $\mathcal{B}^j$. First, after the breakpoint search procedure of Algorithm 1, we obtain the intermediate sets $\tilde{\mathcal{A}}^j$ and $\tilde{\mathcal{B}}^j$ that contain all breakpoints in $\mathcal{A}^j$ and $\mathcal{B}^j$ respectively that have not been considered as candidate breakpoints during the search procedure. Second, the new breakpoint sets $\mathcal{A}^{j+1}$ and $\mathcal{B}^{j+1}$ are obtained by adding to $\tilde{\mathcal{A}}^j$ and $\tilde{\mathcal{B}}^j$ respectively the initial breakpoints α_{j+1}^{j+1} and β_{j+1}^{j+1} corresponding to the $(j+1)$th subproblems. Moreover, we must add any value of the new breakpoints α_i^{j+1} and β_i^{j+1} with $i < j+1$ that is not yet present in these sets. For this, based on the analysis in this subsection, we conclude that it is sufficient to add κ^j to $\tilde{\mathcal{A}}^j$ if there exists $i \in \mathcal{N}^j$ such that $\alpha_i^j < \kappa^j < \beta_i^j$ and to add λ^j to $\tilde{\mathcal{B}}^j$ if there exists $i \in \mathcal{N}^j$ such that $\alpha_i^j < \lambda^j < \beta_i^j$ (see also Table 4). In particular, this means that updating the breakpoint steps can be done in $O(n)$ steps, i.e., by $O(n)$ additions and removals of breakpoint values.

Table 4
Clarification of the four cases used to determine α_i^{j+1}.

Value of α_i^j	Value of β_i^j	Covered by	Resulting value of α_i^{j+1}
$\alpha_i^j < \kappa^j$	$\beta_i^j \leq \kappa^j$	Case 3	β_i^{j+1}; already not present in $\tilde{\mathcal{A}}^j$
$\alpha_i^j < \kappa^j$	$\beta_i^j > \kappa^j$	Case 2	κ^j
$\kappa^j \leq \alpha_i^j \leq \lambda^j$	Any	Case 1	α_i^j
$\lambda^j < \alpha_i^j$	Any	Case 4	α_i^j; already not present in $\tilde{\mathcal{A}}^j$

4.2.3. Updating bookkeeping parameters

In order to efficiently compute the sums $z^j[\delta] := \sum_{i \in \mathcal{N}^j} x_i^j[\delta]$ for a given breakpoint δ, we define the following bookkeeping parameters analogously to those in the breakpoint search procedure for QRAP in Algorithm 1:

$$P^j(\delta) := \sum_{i \leq j:\ \delta < \frac{\bar{l}_i^j}{a_i}} \bar{l}_i^j + \sum_{i \leq j:\ \delta \geq \frac{\bar{u}_i^j}{a_i}} \bar{u}_i^j; \qquad Q^j(\delta) := \sum_{i \leq j:\ \frac{\bar{l}_i^j}{a_i} \leq \delta < \frac{\bar{u}_i^j}{a_i}} a_i;$$

Each breakpoint value $\kappa^{j'}$ and $\lambda^{j'}$ in a given breakpoint set acts as a collective breakpoint for one or multiple activities. As a consequence, within the breakpoint search procedure, they have the same function as the "regular" initial lower and upper breakpoint values α_i^j and β_i^j. Thus, when a breakpoint value of the form $\kappa^{j'}$ or $\lambda^{j'}$ has been considered, we require an efficient update of the bookkeeping sums $P^j(\kappa^{j'})$, $Q^j(\kappa^{j'})$ or $P^j(\lambda^{j'})$, $Q^j(\lambda^{j'})$ respectively. In the case of $\kappa^{j'}$, we update $P^j(\kappa^{j'})$ by subtracting from this value the sum of the lower bounds $\bar{l}_i^j$ of those activities i whose lower breakpoint equals $\kappa^{j'}$, i.e., for which $\alpha_i^j = \kappa^{j'}$. The sum of these values is

$$\sum_{i<j:\ \alpha_i^j = \kappa^{j'}} \bar{l}_i^j = \sum_{i<j:\ \alpha_i^j = \kappa^{j'}} a_i \alpha_i^j = \sum_{i<j:\ \alpha_i^j = \kappa^{j'}} a_i \kappa^{j'} = \kappa^{j'} \sum_{i \leq j':\ \alpha_i^{j'} \leq \kappa^{j'} < \beta_i^{j'}} a_i = Q^{j'}(\kappa^{j'}) \kappa^{j'},$$

since $\alpha_i^j = \frac{\bar{l}_i^j}{a_i}$ for each $i \in \mathcal{N}^j$ and we have that $\alpha_i^j = \kappa^{j'}$ if and only if $\alpha_i^{j''} = \kappa^{j'}$ for all $j'' \in \{j', \dots, j\}$. Analogously, we update the bookkeeping sum $Q^j(\kappa^{j'})$ by adding to this value the sum of the parameters a_i for those i with $\alpha_i^j = \kappa^{j'}$. This sum is

$$\sum_{i<j:\ \alpha_i^j = \kappa^{j'}} a_i = \sum_{i \leq j':\ \alpha_i^{j'} \leq \kappa^{j'} < \beta_i^{j'}} a_i = Q^{j'}(\kappa^{j'}).$$

Thus, the updates take the form $P^j(\kappa^{j'}) - Q^{j'}(\kappa^{j'})\kappa^{j'}$ and $Q^j(\kappa^{j'}) + Q^{j'}(\kappa^{j'})$.

The updates for the case of $\lambda^{j'}$, i.e., for $P^j(\lambda^{j'})$ and $Q^j(\lambda^{j'})$, are analogous to those for the case of $\kappa^{j'}$. Table 5 provides an overview of the updates of the bookkeeping sums for both these cases for each of the four breakpoint values types α_i^j, β_i^j, $\kappa^{j'}$, and $\lambda^{j'}$.

4.3. Recovering the optimal solution to QRAP-NC

In the previous section, we found an efficient way to compute the optimal Lagrange multipliers κ^j and λ^j for the QRAP subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$. In this section, we show how we can use these values to compute the optimal solution $\boldsymbol{x}^n(R)$. For this, we first determine which nested constraints are tight in $x^n(R)$ and use this information to reconstruct the individual terms $x_i^n(R)$ for $i \in \mathcal{N}$. To this end, for each $j \in \mathcal{N}^{n-1}$, let υ^j denote the smallest index larger than or equal to j such that one of its corresponding nested constraints is tight in $\boldsymbol{x}^n(R)$. More precisely,

$$\upsilon^j := \min\left(k \geq j \;\middle|\; \sum_{i \in \mathcal{N}^k} x_i^n(R) = L^k \text{ or } \sum_{i \in \mathcal{N}^k} x_i^n(R) = U^k \right).$$

Furthermore, let V^j denote the value of the tight nested constraint corresponding to the index υ^j and χ^j the corresponding multiplier, i.e., $V^j \in \{L^{\upsilon^j}, U^{\upsilon^j}\}$ and $\chi^j \in \{\kappa^{\upsilon^j}, \lambda^{\upsilon^j}\}$. More precisely,

- $\sum_{i \in \mathcal{N}^{\upsilon^j}} x_i^n(R) = L^{\upsilon^j}$ implies $V^j = L^{\upsilon^j}$ and $\chi^j = \kappa^{\upsilon^j}$;
- $\sum_{i \in \mathcal{N}^{\upsilon^j}} x_i^n(R) = U^{\upsilon^j}$ implies $V^j = U^{\upsilon^j}$ and $\chi^j = \lambda^{\upsilon^j}$.

The main result in this subsection is that the values χ^j act as optimal Lagrange multipliers for the resource constraint (6) in the subproblem QRAP$^n(R)$. As a consequence, given these values, we can calculate $\boldsymbol{x}^n(R)$ directly using a relation similar to the Lagrangian relaxation solution in Eq. (3). To show this result, we prove Lemmas 4 and 5. First, Lemma 4 shows how we can iteratively compute χ from the optimal multipliers $\boldsymbol{\kappa}$ and $\boldsymbol{\lambda}$ using a simple recursive relation. Second, Lemma 5 shows how we can calculate $\boldsymbol{x}^n(R)$ from χ using a relation similar to that in Eq. (3).

Lemma 4. *We have $\chi^n = \kappa^n = \lambda^n$. Moreover, for each $j \in \mathcal{N}^{n-1}$, we have:*

1. *$\chi^{j+1} \leq \kappa^j$ implies $\sum_{i \in \mathcal{N}^j} x_i^n(R) = L^j$ and $\chi^j = \kappa^j$;*
2. *$\lambda^j \leq \chi^{j+1}$ implies $\sum_{i \in \mathcal{N}^j} x_i^n(R) = U^j$ and $\chi^j = \lambda^j$,*
3. *$\kappa^j < \chi^{j+1} < \lambda^j$ implies $L^j < \sum_{i \in \mathcal{N}^j} x_i^n(R) < U^j$ and $\chi^j = \chi^{j+1}$.*

Proof. See Appendix A.2. □

Lemma 5. *For each $i \in \mathcal{N}$, we have*

$$x_i^n(R) = \begin{cases} l_i & \text{if } \chi^i < \alpha_i^i, \\ a_i \chi^i & \text{if } \alpha_i^i \leq \chi^i < \beta_i^i, \\ u_i & \text{if } \beta_i^i \leq \chi^i. \end{cases} \tag{11}$$

Proof. See Appendix A.3. □

Note that, starting from $\chi^n = \kappa^n$ and using Lemma 4, we can compute the values χ^j recursively as

$$\chi^j = \begin{cases} \kappa^j & \text{if } \chi^{j+1} \leq \kappa^j, \\ \lambda^j & \text{if } \chi^{j+1} \geq \lambda^j, \\ \chi^{j+1} & \text{otherwise.} \end{cases} \tag{12}$$

Thus, given the optimal Lagrange multipliers $\boldsymbol{\kappa}$ and $\boldsymbol{\lambda}$, we can compute the optimal solution $\bar{\boldsymbol{x}}$ to QRAP-NC in $O(n)$ time as $\boldsymbol{x}^n(R)$ using the two relatively simple recursions in Eqs. (11) and (12).

4.4. An $O(n \log n)$ time algorithm for QRAP-NC

In the previous two subsections, we derived an efficient approach to compute the optimal Lagrange multipliers $\boldsymbol{\kappa}$ and $\boldsymbol{\lambda}$ for the QRAP$^j(L^j)$ and QRAP$^j(U^j)$ subproblems and to compute from these multipliers the optimal solution $\bar{\boldsymbol{x}}$. In this subsection, we combine these two ingredients to formulate a fast and efficient algorithm for QRAP-NC (Algorithm 3). More precisely, in the first part of this subsection, Section 4.4.1, we present our algorithm and discuss several of its details regarding the subroutines for computing the optimal Lagrange multipliers of the QRAP$^j(L^j)$ and QRAP$^j(U^j)$ subproblems. This includes several procedures that deal with corner cases and with the updating of the breakpoint sets and the bookkeeping parameters. In the second part, Section 4.4.2, we focus on the efficiency of the algorithm. In particular, we prove in Lemma 6 that the algorithm has an $O(n \log n)$ worst-case time complexity when using an appropriate data structure. Finally, in the third part, we compare the complexities of the three algorithms ALG_{seq}, ALG_{inf}, and ALG_{dec} and present. In particular, we discuss a worst-case instance for ALG_{seq} that is essentially a best-case instance for ALG_{inf} and a worst-case instance for ALG_{inf} that is essentially a best-case instance for ALG_{seq}.

4.4.1. Description of the algorithm

Algorithm 3 captures our approach for solving QRAP-NC. First, in Lines 3–13, the algorithm initializes all problem parameters, the initial breakpoint values and breakpoint sets, and the initial bookkeeping

Table 5
Updating the bookkeeping sums $P^j(\delta)$ and $Q^j(\delta)$ when searching the breakpoints in non-decreasing order (QRAP$^j(L^j)$) and non-increasing order (QRAP$^j(U^j)$).

Type of δ	In QRAP$^j(L^j)$ (non-decreasing search)		In QRAP$^j(U^j)$ (non-increasing search)	
	$P^j(\delta)$	$Q^j(\delta)$	$P^j(\delta)$	$Q^j(\delta)$
$\delta \equiv \alpha_i^j$	$P^j(\delta) - l_i$	$Q^j(\delta)_i + a_i$	$P^j(\delta) + l_i$	$Q^j(\delta) - a_i$
$\delta \equiv \beta_i^j$	$P^j(\delta) + u_i$	$Q^j(\delta) - a_i$	$P^j(\delta) - u_i$	$Q^j(\delta) + a_i$
$\delta \equiv \kappa^{j'},\ j' < j$	$P^j(\delta) - Q^{j'}(\kappa^{j'})\kappa^{j'}$	$Q^j(\delta) + Q^{j'}(\kappa^{j'})$	$P^j(\delta) + Q^{j'}(\kappa^{j'})\kappa^{j'}$	$Q^j(\delta) - Q^{j'}(\kappa^{j'})$
$\delta \equiv \lambda^{j'},\ j' < j$	$P^j(\delta) + Q^{j'}(\lambda^{j'})\lambda^{j'}$	$Q^j(\delta) - Q^{j'}(\lambda^{j'})$	$P^j(\delta) - Q^{j'}(\lambda^{j'})\lambda^{j'}$	$Q^j(\delta) + Q^{j'}(\lambda^{j'})$

parameters. Throughout the entire algorithm, it maintains four separate sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{K}$, and $\mathcal{L}$ of breakpoint values corresponding to the "source" of the values, i.e., this specifies whether they are one of the initial breakpoint values α_i^i or β_i^i or one of the optimal Lagrange multipliers κ^j or λ^j respectively. Second, in Lines 14–16, the algorithm applies for each $j \in \mathcal{N}\backslash\{1\}$ the procedure SolveSubproblems(j) (see Algorithm 4) that computes the optimal Lagrange multipliers κ^j and λ^j for the two subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$. Finally, using the obtained vectors of optimal Lagrange multipliers κ and λ, the algorithm computes in Lines 17–22 the (alternative) multiplier values χ using the recursion in Eq. (12) and from these values the solution $\boldsymbol{x}^n(R)$ using Eq. (11).

Algorithm 3 An $O(n \log n)$ time algorithm for QRAP-NC.

1: **Input:** Parameters $\boldsymbol{a} \in \mathbb{R}^n_{>0}$, $\boldsymbol{L}, \boldsymbol{U} \in \mathbb{R}^{n-1}$, $R \in \mathbb{R}$, and $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$
2: **Output:** Optimal solution $\bar{\boldsymbol{x}}$ to QRAP-NC
3: $L^1 = l_1 = \max(L^1, l_1)$; $U^1 = u_1 = \min(U^1, u_1)$
4: **for** $j = 2$ to n **do**
5: $\quad L^j = \max(L^j, L^{j-1} + l_j)$
6: $\quad U^j = \min(U^j, U^{j-1} + u_j)$
7: **end for**
8: **for** $i = 1$ to n **do**
9: $\quad \alpha_i^i = \frac{l_i}{a_i}$; $\beta_i^i = \frac{u_i}{a_i}$
10: **end for**
11: $\kappa^1 = \alpha_1^1$; $\lambda^1 = \beta_1^1$; $\kappa^j = \infty$, $\lambda^j = -\infty$ for $j > 1$
12: Initialize breakpoint sets: $\mathcal{A} := \{\alpha_1^1\}$; $\mathcal{B} := \{\beta_1^1\}$; $\mathcal{K} := \emptyset$; $\mathcal{L} := \emptyset$
13: Initialize bookkeeping sums: $\bar{P}_L^1 = \bar{P}_U^1 = 0$; $\bar{Q}_L^1 = \bar{Q}_U^1 = a_1$
14: **for** $j = 2$ to n **do**
15: $\quad$ Apply procedure SolveSubproblems(j)
16: **end for**
17: $\chi^n := \kappa^n$
18: **for** $i = n - 1$ down to 1 **do**
19: $\quad$ Compute χ^i using Equation (12)
20: **end for**
21: Compute $\boldsymbol{x}^n(R)$ using Equation (11)
22: **return** Optimal solution $\bar{\boldsymbol{x}} := \boldsymbol{x}^n(R)$

The procedure SolveSubproblems(j) carries out the breakpoint search procedure for the subproblems QRAP$^j(L^j)$ and QRAP$^j(U^j)$ as described in Section 2 (Lines 39–59). This is done by first initializing the bookkeeping parameters for these breakpoint search procedures in Lines 39–46 and Lines 49–56 and subsequently applying the procedures LowerSubproblem(j) (Line 47, Algorithm 5) and UpperSubproblem(j) (Line 57, Algorithm 6), which are identical in nature to Lines 5–22 of Algorithm 1. Before carrying out the breakpoint search procedure, two possible corner cases are considered in Lines 1–38 with regard to relation between the to-be-computed multipliers κ^j and λ^j and their predecessors κ^{j-1} and λ^{j-1}. We briefly discuss these corner cases for κ^j; the corner cases for λ^j are analogous.

The first corner case occurs when $\kappa^j = \kappa^{j-1}$ (Lines 1–9 in SolveSubproblems(j)). This case corresponds to Lines 7–9 in Algorithm 1, where the currently considered candidate multiplier δ_i leads to a solution $x[\delta_i]$ that sums to C, i.e., $z[\delta_i] = C$. For QRAP-NC$^j(L^j)$, this case thus occurs if and only if $L^{j-1} + x_j^j[\kappa^j] = L^j$, i.e., if and only if $x_i^j(L^j) = x_i^{j-1}(L^{j-1})$ for all $i \in \mathcal{N}^{j-1}$ and $L^{j-1} + \max(l_j, \min(a_j\chi^j, u_j)) = L^j$. The second case (Lines 10–13) occurs when $\kappa^j < \kappa^{j-1}$ and corresponds to Lines 10–12 of Algorithm 1, where the candidate multiplier δ_i leads to a solution $\boldsymbol{x}[\delta_i]$ whose sum is larger than C, i.e., $z[\delta_i] > C$. In QRAP-NC$^j(L^j)$, this case occurs if and only if $L^{j-1} + x_j^j[\kappa^j] > L^j$, i.e., if and only if $x_i^j(L^j) = x_i^{j-1}(L^{j-1})$ for all $i \in \mathcal{N}^{j-1}$ and $L^{j-1} + \max(l_j, \min(a_j\chi^j, u_j)) > L^j$. In both cases, it is not necessary to carry out the actual breakpoint search to find κ^j since either $\kappa^j = \kappa^{j-1}$ (the first case) or $\kappa^j = (L^j - L^{j-1})/a_j$ (the second case).

Whether or not one of the above mentioned corner cases occurs partly determines whether or not we have to include the new initial breakpoint values α_j^j and β_j^j in the breakpoint search procedure. The algorithm makes this decision in Lines 33–38: α_j^j and β_j^j are included only if they are in between the lowest and highest breakpoint values that can be considered in the breakpoint search. This lowest value is κ^j if $\kappa^j \leq \kappa^{j-1}$ (when one of the two corner cases for κ^j occurs and thus this value has already been determined) and κ^{j-1} otherwise (when breakpoint search is required to find κ^j). Analogously, the highest value is λ^j if $\lambda^j \geq \lambda^{j-1}$ and λ^{j-1} otherwise.

4.4.2. Time complexity

We now establish the worst-case time complexity of Algorithm 3 by means of the following lemma:

Lemma 6. *Algorithm 3 can be implemented such that its worst-case time complexity is $O(n \log n)$.*

Proof. Observe that, throughout the algorithm and all its procedures, all operations have a total time complexity of $O(n)$ except for four operations on the sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{K}$, and $\mathcal{L}$ of to-be-considered breakpoints. For each of these breakpoint sets, say D, these are finding the minimum and maximum breakpoint in D (Lines 2 and 18 in Algorithm 4 and Line 2 in Algorithms 5 and 6), inserting a breakpoint value in D (Lines 13, 29, 34, and 37 in Algorithm 4), and removing the minimum or maximum breakpoint from D (Lines 15 and 31 in Algorithm 4 and Lines 16, 20, 24, and 28 in Algorithms 5 and 6). As we showed in Section 4.2, each breakpoint value is inserted and removed at most once during the course of the algorithm. Moreover, in the worst case, we have to find the minimum and maximum breakpoint value in D a number of n times. Thus, the total number of breakpoint set operations is $O(n)$. If we maintain the breakpoint sets as min–max heaps (Atkinson et al., 1986), each of these operations can be executed in $O(1)$ (finding the minimum and maximum) and $O(\log n)$ (inserting and removing a breakpoint) time. This means that the total time complexity of all four breakpoint set operations is $O(n \log n)$ if we use min–max heaps to store the breakpoint sets. It follows that Algorithm 3 can be implemented such that its worst-case time complexity is $O(n \log n)$. □

We expect the average-case performance of Algorithm 3 to be much better than $O(n \log n)$ since the size of the breakpoint sets is limited by the iteration index j. More precisely, in iteration j, the size of each breakpoint set $\mathcal{A}$, $\mathcal{B}$, $\mathcal{K}$, and $\mathcal{L}$ is at most j. As a consequence, the time complexity of insertion and removal of a breakpoint during iteration j is $O(\log j)$ instead of $O(\log n)$.

In practice, carrying out the breakpoint set operations might be faster if we use a different data structure than min–max heaps to

Algorithm 4 Procedure SolveSubproblems(j).

if $L^{j-1} + \max(l_j, \min(a_j\kappa^{j-1}, u_j)) = L^j$ **then** $\{\kappa^j = \kappa^{j-1}\}$
  Replace κ^{j-1} in $\mathcal{K}$ by κ^j
  if $\kappa^j < \alpha_j^j$ **then**
    $\bar{P}_L^j := \bar{P}_L^{j-1} + l_j$; $\bar{Q}_L^j := \bar{Q}_L^{j-1}$
  else if $\beta_j^j < \kappa^j$ **then**
    $\bar{P}_L^j = \bar{P}_L^{j-1} + u_j$; $\bar{Q}_L^j := \bar{Q}_L^{j-1}$
  else
    $\bar{P}_L^j := \bar{P}_L^{j-1}$; $\bar{Q}_L^j := \bar{Q}_L^{j-1} + a_j$
  end if
else if $L^{j-1} + \max(l_j, \min(a_j\kappa^{j-1}, u_j)) > L^j$ **then** $\{\kappa^j < \kappa^{j-1}\}$
  $\kappa^j = (L^j - L^{j-1})/a_j$
  $\bar{P}_L^j := L^{j-1}$; $\bar{Q}_L^j := a_j$
  Add κ^j to $\mathcal{K}$
else $\{\kappa^j > \kappa^{j-1}\}$
  Remove κ^{j-1} from $\mathcal{K}$
end if
if $U^{j-1} + \max(l_j, \min(a_j\lambda^{j-1}, u_j)) = U^j$ **then** $\{\lambda^j = \lambda^{j-1}\}$
  Replace λ^{j-1} in $\mathcal{L}$ by λ^j
  if $\lambda^j < \alpha_j^j$ **then**
    $\bar{P}_U^j := \bar{P}_U^{j-1} + l_j$, $\bar{Q}_U^j := \bar{Q}_U^{j-1}$
  else if $\beta_j^j < \lambda^j$ **then**
    $\bar{P}_U^j = \bar{P}_U^{j-1} + u_j$; $\bar{Q}_U^j := \bar{Q}_U^{j-1}$
  else
    $\bar{P}_U^j := \bar{P}_U^{j-1}$; $\bar{Q}_U^j := \bar{Q}_U^{j-1} + a_j$
  end if
else if $U^{j-1} + \max(l_j, \min(a_j\lambda^{j-1}, u_j)) < U^j$ **then** $\{\lambda^j > \lambda^{j-1}\}$
  $\lambda^j = (U^j - U^{j-1})/a_j$
  $\bar{P}_U^j := U^{j-1}$; $\bar{Q}_U^j := a_j$
  Add λ^j to $\mathcal{L}$
else $\{\lambda^j < \lambda^{j-1}\}$
  Remove λ^{j-1} from $\mathcal{L}$
end if
if $\min(\kappa^{j-1}, \kappa^j) < \alpha_j^j \le \max(\lambda^{j-1}, \lambda^j)$ **then**
  Add α_j^j to $\mathcal{A}$
end if
if $\min(\kappa^{j-1}, \kappa^j) \le \beta_j^j < \max(\lambda^{j-1}, \lambda^j)$ **then**
  Add β_j^j to $\mathcal{B}$
end if
if $\kappa^j > \kappa^{j-1}$ **then**
  if $\kappa^{j-1} < \alpha_j^j$ **then**
    $P := \bar{P}_L^{j-1} + l_j$; $Q := \bar{Q}_L^{j-1}$
  else if $\alpha_j^j \le \kappa^{j-1} < \beta_j^j$ **then**
    $P := \bar{P}_L^{j-1}$; $Q := \bar{Q}_L^{j-1} + a_j$
  else
    $P := \bar{P}_L^{j-1} + u_j$; $Q := \bar{Q}_L^{j-1}$
  end if
  Apply procedure LowerSubproblem(j)
end if
if $\lambda^j < \lambda^{j-1}$ **then**
  if $\beta_j^j < \lambda^{j-1}$ **then**
    $P := \bar{P}_U^{j-1} + u_j$; $Q := \bar{Q}_U^{j-1}$
  else if $\alpha_j^j < \lambda^{j-1} \le \beta_j^j$ **then**
    $P := \bar{P}_U^{j-1}$; $Q := \bar{Q}_U^{j-1} + a_j$
  else
    $P := \bar{P}_U^{j-1} + l_j$; $Q := \bar{Q}_U^{j-1}$
  end if
  Apply procedure UpperSubproblem(j)
end if

maintain the breakpoint sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{K}$, and $\mathcal{L}$. For instance, when n is small, simple arrays might be sufficient for fast insertion and removal of breakpoints, even though this increases the worst-case time complexity to $O(n^2)$. On the other hand, Hochbaum and Hong (1995) suggest to keep the breakpoint sets by means of a so-called disjoint set data structure (see, e.g., Cormen et al. (2009)). Using such a structure, a sequence of $O(n)$ breakpoint insertions and deletions in sets of size at most n can be done in $O(n)$ time using the algorithm in Gabow and Tarjan (1985). However, it is unclear whether the algorithm in Gabow and Tarjan (1985) is fast in practice for two reasons. First, it is complicated and cumbersome to implement compared to other algorithms for insertion and removal operations on disjoint set data structures (Galil and Italiano, 1991). Second, although the authors mention in a preliminary study (Gabow and Tarjan, 1983) that their algorithm outperforms the state-of-the-art at that moment, the literature contains hardly if any studies on its practical performance. Alternatively, one could use other algorithms (e.g., those evaluated in Patwary et al. (2010)) that have a worse worst-case time complexity but have been shown to be fast in practice.

4.4.3. A comparison of worst-case instances

To compare the difference in performance between ALG_{seq}, ALG_{inf}, and ALG_{dec}, we discuss in this section one worst-case instance for ALG_{seq} and one worst-case instance for ALG_{inf}. In fact, we show that this worst-case instance for ALG_{seq} is essentially a best-case instance for ALG_{inf} and that the discussed worst-case instance for ALG_{inf} is a best-case instance for ALG_{seq}. This suggests that the algorithms are in a sense complementary. In general, these analyses may provide additional insights into the strengths and weaknesses of these algorithms.

First, we discuss a worst-case instance for ALG_{seq}. From the time complexity analysis for ALG_{seq} in Section 4.4.2, we may conclude that a worst-case instance for ALG_{seq} is one where the maximum size of the breakpoint set $\mathcal{D}$ is $\Omega(n)$. In particular, if for a given instance first all breakpoints are inserted in and afterwards removed from $\mathcal{D}$, the time complexity all breakpoint operations together is $\sum_{i\in\mathcal{N}} \Omega(\log i) = \Omega(n \log n)$. This holds, e.g., for the following instance:

$$
\begin{aligned}
\text{WC-1:} \quad \min_{x\in\mathbb{R}^n} \ & \sum_{i\in\mathcal{N}} \frac{1}{2}x_i^2 \\
\text{s.t.} \ & \sum_{i\in\mathcal{N}} = 2n^2 + n - \frac{1}{2}n(n+1) - \frac{1}{2}, \\
& \frac{1}{2}j(j+1) \le \sum_{i\in\mathcal{N}^j} x_i \le 2jn + j - \frac{1}{2}j(j+1), \quad j \in \mathcal{N}^{n-1}, \\
& i \le x_i \le 2n + 1 - i, \quad i \in \mathcal{N}.
\end{aligned}
$$

Note that this is in fact an instance of QRAP since the nested constraints are redundant, i.e., we have $\sum_{i\in\mathcal{N}^j} i = \frac{1}{2}j(j+1)$ and $\sum_{i\in\mathcal{N}^j}(2n+1-i) = 2jn + j - \frac{1}{2}j(j+1)$ for all $j \in \mathcal{N}^{n-1}$. We show by means of Lemma 7 that WC-1 is indeed a worst-case instance of ALG_{seq}:

Lemma 7. *When applying ALG_{seq} to WC-1, first $\Omega(n)$ breakpoints are inserted in the breakpoint set $\mathcal{D}$ and subsequently $\Omega(n)$ breakpoints are removed from $\mathcal{D}$ throughout the course of the algorithm.*

Proof. See Appendix A.4. □

Considering the performance of the two other algorithms for this instance, note that the time complexity of ALG_{inf} reduces to $O(n)$ since the nested constraints are redundant and thus not violated in the

Algorithm 5 Procedure LowerSubproblem(j).

1: **repeat**
2: Choose minimum to-be-considered breakpoint: $\delta := \min(\mathcal{A}, \mathcal{B}, \mathcal{K}, \mathcal{L})$ and corresponding member set $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}, \mathcal{K}, \mathcal{L}\}$
3: **if** $P + Q\delta = L^j$ **then**
4: $\kappa^j := \delta$; add κ^j to $\mathcal{K}$
5: $\bar{P}_L^j := P$, $\bar{Q}_L^j := Q$
6: **return**
7: **else if** $P + Q\delta > L^j$ **then**
8: ($\kappa^j < \delta$:) $\kappa^j := (L^j - P)/Q$; add κ^j to $\mathcal{K}$
9: $\bar{P}_L^j := P$, $\bar{Q}_L^j := Q$
10: **return**
11: **else**
12: ($\kappa^j > \delta$:) breakpoint δ will be considered
13: **if** $\mathcal{D} \equiv \mathcal{A}$ **then**
14: Let breakpoint be $\delta \equiv \alpha_k^k$
15: $P := P - l_k$; $Q := Q + a_k$
16: Remove α_k^k from $\mathcal{A}$
17: **else if** $\mathcal{D} \equiv \mathcal{B}$ **then**
18: Let breakpoint be $\delta \equiv \beta_k^k$
19: $P := P + u_k$; $Q := Q - a_k$
20: Remove β_k^k from $\mathcal{B}$
21: **else if** $\mathcal{D} \equiv \mathcal{K}$ **then**
22: Let breakpoint be $\delta \equiv \kappa^k$
23: $P := P - \bar{Q}_L^k \kappa^k$, $Q := Q + \bar{Q}_L^k$
24: Remove κ^k from $\mathcal{K}$
25: **else**
26: Let breakpoint be $\delta \equiv \lambda^k$
27: $P := P + \bar{Q}_U^k \lambda^k$; $Q := Q - \bar{Q}_U^k$
28: Remove λ^k from $\mathcal{L}$
29: **end if**
30: **end if**
31: **until** κ^j has been determined

Algorithm 6 Procedure UpperSubproblem(j).

1: **repeat**
2: Choose maximum to-be-considered breakpoint: $\delta := \min(\mathcal{A}, \mathcal{B}, \mathcal{K}, \mathcal{L})$ and corresponding member set $\mathcal{D} \in \{\mathcal{A}, \mathcal{B}, \mathcal{K}, \mathcal{L}\}$
3: **if** $P + Q\delta = U^j$ **then**
4: $\lambda^j := \delta$; add λ^j to $\mathcal{L}$
5: $\bar{P}_U^j := P$, $\bar{Q}_U^j := Q$
6: **return**
7: **else if** $P + Q\delta < U^j$ **then**
8: ($\lambda^j > \delta$:) $\lambda^j := (U^j - P)/Q$; add λ^j to $\mathcal{L}$
9: $\bar{P}_U^j := P$, $\bar{Q}_U^j := Q$
10: **return**
11: **else**
12: ($\lambda^j < \delta$): breakpoint will be considered
13: **if** $\mathcal{D} \equiv \mathcal{A}$ **then**
14: Let breakpoint be $\delta \equiv \alpha_k^k$
15: $P := P + l_k$, $Q := Q - a_k$
16: Remove α_k^k from $\mathcal{A}$
17: **else if** $\mathcal{D} \equiv \mathcal{B}$ **then**
18: Let breakpoint be $\delta \equiv \beta_k^k$
19: $P := P - u_k$; $Q := Q + a_k$
20: Remove β_k^k from $\mathcal{B}$
21: **else if** $\mathcal{D} \equiv \mathcal{K}$ **then**
22: Let breakpoint be $\delta \equiv \kappa^k$
23: $P := P + \bar{Q}_L^k \kappa^k$, $Q := Q - \bar{Q}_L^k$
24: Remove κ^k from $\mathcal{K}$
25: **else**
26: Let breakpoint be $\delta \equiv \lambda^k$
27: $P := P - \bar{Q}_U^k \lambda^k$; $Q := Q + \bar{Q}_U^k$
28: Remove λ^k from $\mathcal{L}$
29: **end if**
30: **end if**
31: **until** λ^j has been determined

relaxation of the problem. Furthermore, the time complexity of ALG_{dec} remains $O(n \log n)$ since the parameter choices do not influence the level of recursion and the number of subproblems that must be solved at each level of the recursion hierarchy.

Now we discuss a worst-case instance for ALG_{inf}. This instance is described in Wu et al. (2021) under the additional requirement that the decision variables are integer-valued. However, the conclusion of their analysis, namely that the instance is a worst-case instance, is not affected by this requirement. As a consequence, it is also a worst-case instance of QRAP-NC with continuous variables for ALG_{inf}.

The instance is given by

$$\begin{aligned}
\text{WC-2:} \quad \min_{x \in \mathbb{R}^n} \ & \sum_{i \in \mathcal{N}} \frac{1}{2} x_i^2 \\
\text{s.t.} \ & \sum_{i \in \mathcal{N}} x_i = (-1)^n n, \\
& (-1)^j j \leq \sum_{i \in \mathcal{N}^j} x_i \leq (-1)^j j + 1, \quad j \in \mathcal{N}^{n-1}, \\
& -2n \leq x_i \leq 2n, \quad i \in \mathcal{N}.
\end{aligned}$$

In the following, we determine for this instance the number of breakpoint operations and the size of the breakpoint sets in ALG_{seq}. For this particular instance, we can derive a closed-form expression for the multipliers of the QRAP subproblems, i.e., $\boldsymbol{\kappa}$ and $\boldsymbol{\lambda}$:

Lemma 8. *For the instance WC-2, the Lagrange multipliers κ^j and λ^j as computed by ALG_{seq} are given for each $j \in \mathcal{N}^{n-1}$ by*

$$\kappa^j := (-1)^j \left(2j - \frac{3}{2}\right) - \frac{1}{2}, \qquad \lambda^j := (-1)^j \left(2j - \frac{3}{2}\right) + \frac{1}{2}.$$

Proof. See Appendix A.5. □

Note that Lemma 8 implies that $\kappa^j < \kappa^{j-1}$ and $\lambda^j < \lambda^{j-1}$ for j uneven and $\kappa^j > \kappa^{j-1}$ and $\lambda^j > \lambda^{j-1}$ for j even. Thus, in procedure SolveSubproblems(j), either κ^j is added to $\mathcal{K}$ (Line 13) and λ^{j-1} is removed from $\mathcal{L}$ (Line 31) or κ^{j-1} is removed from $\mathcal{K}$ (Line 15) and λ^j is added to $\mathcal{L}$ (Line 29). Moreover, since $\kappa^j > -2n = \alpha_j^j$ and $\lambda^j < 2n = \beta_j^j$ for all $j > 2$, none of the initial breakpoints α_j^j and β_j^j are added to $\mathcal{A}$ and $\mathcal{B}$ in Lines 33–38. Together, this means that the maximum number of breakpoints in $\mathcal{D} = \mathcal{A} \cup \mathcal{B} \cup \mathcal{K} \cup \mathcal{L}$ at any moment during the course of the algorithm is at most 3 (the initial number of breakpoints in $\mathcal{D}$ plus one). Thus, the maximum required depth of the heaps that store the breakpoint sets is independent of the problem size n. This means that also the efficiency of each individual breakpoint operation on these heaps is $O(1)$ and thus the time complexity of all operations together reduces from $O(n \log n)$ to $O(n)$. Since all other operations of ALG_{seq} have a total time complexity of $O(n)$, the time complexity of ALG_{seq} for the considered instance is $O(n)$.

We conclude this section with a final remark on the applicability of ALG_{seq} to the case where the variables are integer-valued. At the end of Section 3, we mentioned that the initial sequential Algorithm 2 can be extended to integer-valued decision variables. The reason for this is that, in this algorithm, the QRAP subproblems are solved explicitly. Thus, these intermediate solutions can be used directly as lower and upper single-variable bounds for the next subproblem. In ALG_{seq}, the unique optimal solution to each subproblem is not explicitly computed. Instead, we compute the (unique) Lagrange multiplier corresponding to this solution. However, in the problem with integer variables, subproblems may not have unique optimal solutions. Moreover, the characterizing property of Lagrange multipliers does not hold anymore. This means that there may be multiple optimal solutions that are characterized by the same Lagrange multiplier or such a multiplier

Table 6
Parameter choices for the battery scheduling problem for each scenario.

	$X_{\min}$	$X_{\max}$	D
Small	$-4.0 \cdot 10^3$	$4.0 \cdot 10^3$	$8.0 \cdot 10^4$
Medium	$-2.0 \cdot 10^4$	$2.0 \cdot 10^4$	$4.0 \cdot 10^5$
Large	$-3.6 \cdot 10^4$	$3.6 \cdot 10^4$	$7.2 \cdot 10^5$

might not even exist. Since the uniqueness of the optimal solutions and the presented characterization of solutions via Lagrange multipliers play a crucial role in ALG_{seq}, we believe that a direct extension of this algorithm to integer-valued decision variables might not be possible.

5. Evaluation

In this section, we evaluate the performance of our Algorithm 3 as presented in Section 4.4, to which we shall refer as ALG_{seq} for clarity, and compare it with the state-of-the-art algorithms ALG_{inf} from van der Klauw et al. (2017) and ALG_{dec} from Vidal et al. (2019). We carry out two types of experiments. First, we evaluate the performance of our algorithm on instances of the battery scheduling problem BATTERY using measured power consumption data as input. For this, we tailor ALG_{seq} to this problem and compare this implementation to a tailored implementation of ALG_{inf} within the simulation tool DEMKit (Hoogsteen et al., 2019). Second, we compare the execution time and scalability of our algorithm and of ALG_{inf} and ALG_{dec} on synthetic instances with sizes ranging from 10 to ten million variables. We have implemented all three algorithms in Python (version 3.5) to be able to compare them to the implementation in DEMKit, which is also written in Python, and made the corresponding code available at https://github.com/mhhschootuiterkamp/QRAP_NC. All simulations and computations have been executed on a 2.60 GHz Dell Inspiron 15 with an Intel Core i7-6700HQ CPU and 16 GB of RAM.

In Section 5.1, we describe in more detail the problem instances that we use in the evaluation. Subsequently, in Section 5.2, we discuss several implementation choices and in Section 5.3 we present and discuss the results of our evaluation.

5.1. Problem instances

For the comparison of the tailored implementation of our algorithm ALG_{seq} with the tailored implementation of ALG_{inf} within DEMKit, we generate instances of the problem BATTERY using measured power consumption data as input. For this, we consider the setting where a battery charging schedule for two consecutive days needs to be computed. This scheduling horizon is divided into 15-minute time intervals, resulting in $n = 192$. To study the influence of the battery size on the solving time, we consider three scenarios that correspond to three different battery sizes and denote them by Small, Medium, and Large. In these scenarios, the battery capacity is 20 kWh, 100 kWh, or 180 kWh and the (dis)charging rate is 4 kW, 20 kW, or 36 kW respectively. This leads to $\Delta t = \frac{1}{4}$ and to the values for $X_{\min}$, $X_{\max}$, and D as given in Table 6. Note that this is equivalent to the situation where either 10, 50, or 90 percent of the households have installed a smaller "home" battery with a capacity of 5 kWh and a (dis)charging rate of 1 kW, which corresponds to real-life field tests such as described in Reijnders et al. (2018). We set both the initial and target SoC to a given fraction of the capacity, i.e., $S_{\text{start}} = S_{\text{end}} = sD$, where $s \in \{0, 0.1, 0.2, \dots, 1\}$. For each scenario, we simulate 50 battery schedules of two days. As input for the base load $\boldsymbol{p}$, we use measurement data of the actual power consumption of 40 households for 100 consecutive days that were obtained in the field test described in Hoogsteen et al. (2017).

For the scalability analysis, we generate synthetic instances in the same way as in Vidal et al. (2019). For this, we consider instance sizes n in the set $\{10, 20, 50, 100, 200, 500, \dots, 10^7\}$ and for each of these sizes, we generate 10 instances. In each instance, we sample the parameters a, l, and u from the uniform distributions $U(0, 1)$, $U(0.1, 0.5)$, and $U(0.5, 0.9)$ respectively. To generate the nested bounds L and U, we first draw for each $i \in \mathcal{N}$ two values X_i and Y_i from the uniform distribution $U(l_i, u_i)$. Subsequently, we define for each $j \in \mathcal{N}$ the values $w_1^j := \sum_{i \in \mathcal{N}^j} X_i$ and $w_2^j := \sum_{i \in \mathcal{N}^j} Y_i$ and we set $L^j := \min(w_1^j, w_j^2)$ and $U^j := \max(w_1^j, w_2^j)$ for $j < n$ and $L^n = U^n = \frac{1}{2}(w_1^n + w_2^n)$.

5.2. Implementation details

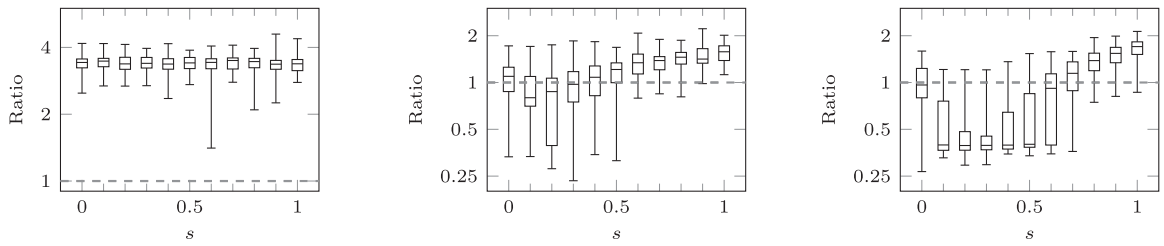
In both the divide-and-conquer algorithm ALG_{dec} and the infeasibility-guided algorithm ALG_{inf}, we use the algorithm from Kiwiel (2008) to solve the QRAP subproblems. In this algorithm, we replaced the proposed linear-time procedure for finding medians by the procedure `statistics.median()`. The reason for this is, as mentioned before, that linear-time algorithms such as in Blum et al. (1973) for finding medians are relatively slow in practice due to the large constant factor in their complexity (see also Kiwiel (2005) and Alexandrescu (2017)).

For the double-ended queues needed in ALG_{seq} for the optimal Lagrange multipliers κ and λ, we use the Python container data type `deque`. Moreover, we initially implemented the double-ended priority queues for the lower and upper initial breakpoint values $(\alpha_i^j)_{i \in \mathcal{N}}$ and $(\beta_i^j)_{i \in \mathcal{N}}$ as symmetric min–max heaps (Arvind and Rangan, 1999). However, initial tests indicated that using instead a coupled min-heap and max-heap implementation with total correspondence leads to similar or even lower execution times of the overall algorithm. Moreover, the latter data structure is much simpler to implement using the standard Python library `heapq`. Therefore, we use this method instead of min–max heaps. In this alternative method, we insert new breakpoints in both the min-heap and the max-heap and use the min-heap to find and delete a minimum breakpoint (in the lower subproblems) and the max-heap to find and delete a maximum breakpoint (in the upper subproblems). Moreover, we assign to each breakpoint a flag that is 1 if the breakpoint has been removed from either of the heaps and 0 otherwise. This prevents that we find a minimum (maximum) breakpoint in the min-heap (max-heap) that was already considered in the other heap and thus has been removed from the breakpoint search.

5.3. Results and discussion

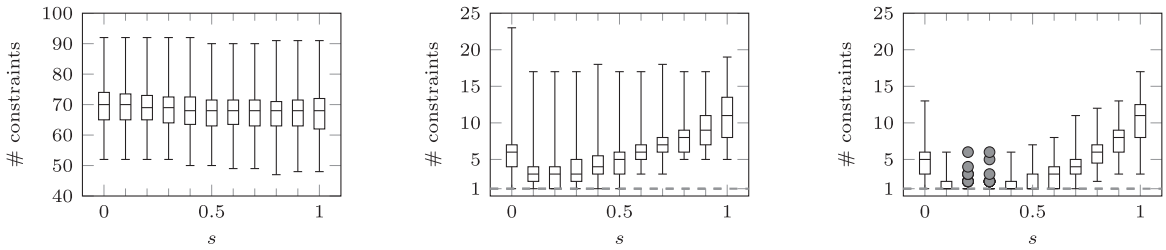
In this section, we present and discuss the results of our evaluation. First, we discuss the results of the comparison of the tailored implementation of ALG_{seq} with the tailored implementation of ALG_{inf} within DEMKit. Fig. 2 shows the ratios between the execution times of the tailored implementation of ALG_{inf} and that of ALG_{seq}. Moreover, Tables 7–9 contain for each scenario and each initial and target SoC value the mean, maximum, and coefficient of variation (CoV) of the execution times. The CoV is the sample deviation divided by the sample mean and is a suitable measure of the variation between samples when comparing different collections of samples with significantly different sample means.

Tables 7–9 show that the mean execution time of ALG_{seq} is similar in each scenario, whereas that of ALG_{inf} appears to decrease as the battery size increases. This implies that also the ratios between the execution times decrease as the battery size increases, which is confirmed by the boxplots in Fig. 2. In particular, a smaller battery size seems to imply that ALG_{seq} is likely to be faster than ALG_{inf} whereas ALG_{inf} is likely to be faster for larger battery size. The reason for this is that the execution time of ALG_{inf} heavily depends on the number of tight nested constraints in an optimal solution, i.e., on the number of distinct values of v^j for $j \in \mathcal{N}$ (see also Section 4.3). To support this fact, we plot in Fig. 3 boxplots of these numbers. Note that when the initial SoC is 20% or 30% of the battery capacity in the scenario Large, in only 4 of the 50 instances the number of tight constraints was more than 1, meaning that in the remaining 46 instances the optimal solution to the relaxation of the problem did not violate any of the nested constraints.



(a) Small. (b) Medium. (c) Large.

Fig. 2. Boxplots of the execution time of the tailored implementation of ALG_{inf} within DEMKit divided by that of the tailored implementation of ALG_{seq} for the three scenarios. Ratios larger than 1 imply that ALG_{seq} was faster than ALG_{inf}.



(a) Small. (b) Medium. (c) Large.

Fig. 3. Boxplots of the number of tight constraints in the optimal solutions for the three scenarios.

Table 7
The mean, maximum, and coefficient of variation of the execution times of the tailored implementation of ALG_{seq} and the tailored implementation of ALG_{inf} within DEMKit for the scenario Small.

s	ALG_{seq}			ALG_{inf} within DEMKit		
	Mean (s)	Max (s)	CoV	Mean (s)	Max (s)	CoV
0	$1.80 \cdot 10^{-3}$	$2.09 \cdot 10^{-3}$	$5.80 \cdot 10^{-2}$	$6.15 \cdot 10^{-3}$	$7.23 \cdot 10^{-3}$	$7.61 \cdot 10^{-2}$
0.1	$1.78 \cdot 10^{-3}$	$2.14 \cdot 10^{-3}$	$6.18 \cdot 10^{-2}$	$6.15 \cdot 10^{-3}$	$7.16 \cdot 10^{-3}$	$6.52 \cdot 10^{-2}$
0.2	$1.81 \cdot 10^{-3}$	$2.77 \cdot 10^{-8}$	$9.19 \cdot 10^{-2}$	$6.10 \cdot 10^{-3}$	$7.15 \cdot 10^{-3}$	$7.17 \cdot 10^{-2}$
0.3	$1.79 \cdot 10^{-3}$	$2.25 \cdot 10^{-3}$	$7.70 \cdot 10^{-2}$	$6.10 \cdot 10^{-3}$	$7.10 \cdot 10^{-3}$	$7.34 \cdot 10^{-2}$
0.4	$1.79 \cdot 10^{-3}$	$2.44 \cdot 10^{-3}$	$7.25 \cdot 10^{-2}$	$6.07 \cdot 10^{-3}$	$7.11 \cdot 10^{-3}$	$7.10 \cdot 10^{-2}$
0.5	$1.77 \cdot 10^{-3}$	$2.25 \cdot 10^{-3}$	$6.28 \cdot 10^{-2}$	$6.02 \cdot 10^{-3}$	$6.94 \cdot 10^{-3}$	$6.41 \cdot 10^{-2}$
0.6	$1.83 \cdot 10^{-3}$	$4.00 \cdot 10^{-3}$	$1.89 \cdot 10^{-1}$	$6.01 \cdot 10^{-3}$	$7.09 \cdot 10^{-3}$	$6.78 \cdot 10^{-2}$
0.7	$1.77 \cdot 10^{-3}$	$2.21 \cdot 10^{-3}$	$6.49 \cdot 10^{-2}$	$6.05 \cdot 10^{-3}$	$6.97 \cdot 10^{-3}$	$7.69 \cdot 10^{-2}$
0.8	$1.80 \cdot 10^{-3}$	$2.68 \cdot 10^{-3}$	$1.00 \cdot 10^{-1}$	$6.05 \cdot 10^{-3}$	$7.06 \cdot 10^{-3}$	$7.52 \cdot 10^{-2}$
0.9	$1.79 \cdot 10^{-3}$	$2.87 \cdot 10^{-3}$	$1.24 \cdot 10^{-1}$	$5.97 \cdot 10^{-3}$	$7.88 \cdot 10^{-3}$	$8.48 \cdot 10^{-2}$
1	$1.79 \cdot 10^{-3}$	$2.18 \cdot 10^{-3}$	$6.02 \cdot 10^{-2}$	$6.01 \cdot 10^{-3}$	$8.22 \cdot 10^{-3}$	$8.81 \cdot 10^{-2}$

Table 8
The mean, maximum, and coefficient of variation of the execution times of the tailored implementation of ALG_{seq} and the tailored implementation of ALG_{inf} within DEMKit for the scenario Medium.

s	ALG_{seq}			ALG_{inf} within DEMKit		
	Mean (s)	Max (s)	CoV	Mean (s)	Max (s)	CoV
0	$1.71 \cdot 10^{-3}$	$2.40 \cdot 10^{-3}$	$1.19 \cdot 10^{-1}$	$1.88 \cdot 10^{-3}$	$3.45 \cdot 10^{-3}$	$3.10 \cdot 10^{-1}$
0.1	$1.64 \cdot 10^{-3}$	$1.96 \cdot 10^{-3}$	$7.39 \cdot 10^{-2}$	$1.45 \cdot 10^{-3}$	$3.31 \cdot 10^{-3}$	$4.47 \cdot 10^{-1}$
0.2	$1.66 \cdot 10^{-3}$	$2.40 \cdot 10^{-3}$	$1.03 \cdot 10^{-1}$	$1.47 \cdot 10^{-3}$	$3.34 \cdot 10^{-3}$	$4.78 \cdot 10^{-1}$
0.3	$1.76 \cdot 10^{-3}$	$3.50 \cdot 10^{-3}$	$1.85 \cdot 10^{-1}$	$1.68 \cdot 10^{-3}$	$3.43 \cdot 10^{-3}$	$4.41 \cdot 10^{-1}$
0.4	$1.65 \cdot 10^{-3}$	$2.36 \cdot 10^{-3}$	$1.07 \cdot 10^{-1}$	$1.76 \cdot 10^{-3}$	$3.51 \cdot 10^{-3}$	$3.45 \cdot 10^{-1}$
0.5	$1.66 \cdot 10^{-3}$	$2.42 \cdot 10^{-3}$	$1.08 \cdot 10^{-1}$	$1.95 \cdot 10^{-3}$	$3.27 \cdot 10^{-3}$	$2.83 \cdot 10^{-1}$
0.6	$1.65 \cdot 10^{-3}$	$2.48 \cdot 10^{-3}$	$9.75 \cdot 10^{-2}$	$2.23 \cdot 10^{-3}$	$3.52 \cdot 10^{-3}$	$2.26 \cdot 10^{-1}$
0.7	$1.68 \cdot 10^{-3}$	$2.09 \cdot 10^{-3}$	$8.46 \cdot 10^{-2}$	$2.32 \cdot 10^{-3}$	$3.42 \cdot 10^{-3}$	$1.85 \cdot 10^{-1}$
0.8	$1.64 \cdot 10^{-3}$	$1.94 \cdot 10^{-3}$	$6.79 \cdot 10^{-2}$	$2.36 \cdot 10^{-3}$	$3.19 \cdot 10^{-3}$	$1.61 \cdot 10^{-1}$
0.9	$1.70 \cdot 10^{-3}$	$2.17 \cdot 10^{-3}$	$1.02 \cdot 10^{-1}$	$2.53 \cdot 10^{-3}$	$3.50 \cdot 10^{-3}$	$1.63 \cdot 10^{-1}$
1	$1.67 \cdot 10^{-3}$	$2.11 \cdot 10^{-3}$	$8.41 \cdot 10^{-2}$	$2.61 \cdot 10^{-3}$	$4.05 \cdot 10^{-3}$	$1.54 \cdot 10^{-1}$

The relation between the number of tight nested constraints and the ratios is also strongly visible when comparing Figs. 2 and 3: the ratios increase as the number of tight constraints increases.

Table 9
The mean, maximum, and coefficient of variation of the execution times of the tailored implementation of ALG_{seq} and the tailored implementation of ALG_{inf} within DEMKit for the scenario Large.

s	ALG_{seq}			ALG_{inf} within DEMKit		
	Mean (s)	Max (s)	CoV	Mean (s)	Max (s)	CoV
0	$1.59 \cdot 10^{-3}$	$2.15 \cdot 10^{-3}$	$7.33 \cdot 10^{-2}$	$1.60 \cdot 10^{-3}$	$2.76 \cdot 10^{-3}$	$2.98 \cdot 10^{-1}$
0.1	$1.58 \cdot 10^{-3}$	$1.90 \cdot 10^{-3}$	$7.33 \cdot 10^{-2}$	$8.93 \cdot 10^{-4}$	$2.11 \cdot 10^{-3}$	$5.22 \cdot 10^{-1}$
0.2	$1.58 \cdot 10^{-3}$	$1.94 \cdot 10^{-3}$	$8.61 \cdot 10^{-2}$	$7.82 \cdot 10^{-4}$	$2.04 \cdot 10^{-3}$	$4.77 \cdot 10^{-1}$
0.3	$1.55 \cdot 10^{-3}$	$2.16 \cdot 10^{-3}$	$8.79 \cdot 10^{-2}$	$7.45 \cdot 10^{-4}$	$2.00 \cdot 10^{-3}$	$4.66 \cdot 10^{-1}$
0.4	$1.54 \cdot 10^{-3}$	$2.05 \cdot 10^{-3}$	$7.06 \cdot 10^{-2}$	$8.22 \cdot 10^{-4}$	$2.30 \cdot 10^{-3}$	$5.18 \cdot 10^{-1}$
0.5	$1.54 \cdot 10^{-3}$	$2.05 \cdot 10^{-3}$	$7.67 \cdot 10^{-2}$	$1.02 \cdot 10^{-3}$	$2.42 \cdot 10^{-3}$	$5.51 \cdot 10^{-1}$
0.6	$1.53 \cdot 10^{-3}$	$1.83 \cdot 10^{-3}$	$6.24 \cdot 10^{-2}$	$1.34 \cdot 10^{-3}$	$2.63 \cdot 10^{-3}$	$4.38 \cdot 10^{-1}$
0.7	$1.55 \cdot 10^{-3}$	$1.77 \cdot 10^{-3}$	$6.22 \cdot 10^{-2}$	$1.72 \cdot 10^{-3}$	$2.61 \cdot 10^{-3}$	$2.75 \cdot 10^{-1}$
0.8	$1.53 \cdot 10^{-3}$	$1.86 \cdot 10^{-3}$	$5.80 \cdot 10^{-2}$	$2.09 \cdot 10^{-3}$	$2.95 \cdot 10^{-3}$	$1.87 \cdot 10^{-1}$
0.9	$1.53 \cdot 10^{-3}$	$1.94 \cdot 10^{-3}$	$6.34 \cdot 10^{-2}$	$2.31 \cdot 10^{-3}$	$2.98 \cdot 10^{-3}$	$1.69 \cdot 10^{-1}$
1	$1.52 \cdot 10^{-3}$	$1.77 \cdot 10^{-3}$	$4.58 \cdot 10^{-2}$	$2.51 \cdot 10^{-3}$	$3.16 \cdot 10^{-3}$	$1.36 \cdot 10^{-1}$

Table 10
Percentage of instances where the tailored implementation of ALG_{seq} is faster than the tailored implementation of ALG_{inf} within DEMKit given the number of tight nested constraints in their optimal solutions.

Number of tight nested constraints	1	2	3	4	5	6	≥ 7
"Win" percentage	0.0	2.2	30.2	62.8	83.8	93.4	100

From these results, we can derive a "rule of thumb" for the choice of a proper algorithm to use given the expected number of tight nested constraints. To this end, we compute for each number of tight constraints the percentage of instances where the tailored implementation of ALG_{seq} runs faster than the tailored implementation of ALG_{inf} within DEMKit given the optimal solution has this particular number of tight nested constraints (see Table 10). These values suggest that when the number of tight constraints is more than $\frac{4}{192} \approx 2.1$ percent, our algorithm is faster in more than 50% of the instances. In particular, when the number of tight constraints is $\frac{7}{192} \approx 3.6$ percent or more, the tailored implementation of our algorithm ALG_{seq} is always faster.

Note that this rule-of-thumb is in line with the physical interpretation of tight nested constraints in BATTERY. For this, note that a battery

Table 11
Mean and coefficient of variation of the execution times.

n	Mean (s)			CoV		
	ALG_{seq}	ALG_{inf}	ALG_{dec}	ALG_{seq}	ALG_{inf}	ALG_{dec}
10	$1.33 \cdot 10^{-4}$	$2.54 \cdot 10^{-4}$	$1.44 \cdot 10^{-3}$	$9.97 \cdot 10^{-1}$	$5.40 \cdot 10^{-1}$	$1.77 \cdot 10^{-1}$
20	$1.80 \cdot 10^{-4}$	$4.12 \cdot 10^{-4}$	$3.11 \cdot 10^{-3}$	$7.32 \cdot 10^{-2}$	$2.32 \cdot 10^{-1}$	$8.01 \cdot 10^{-2}$
50	$5.00 \cdot 10^{-4}$	$1.05 \cdot 10^{-3}$	$9.41 \cdot 10^{-3}$	$3.06 \cdot 10^{-1}$	$2.53 \cdot 10^{-1}$	$2.24 \cdot 10^{-1}$
100	$9.06 \cdot 10^{-4}$	$1.97 \cdot 10^{-3}$	$2.07 \cdot 10^{-2}$	$6.51 \cdot 10^{-2}$	$3.44 \cdot 10^{-1}$	$1.35 \cdot 10^{-1}$
200	$1.90 \cdot 10^{-3}$	$4.02 \cdot 10^{-3}$	$4.16 \cdot 10^{-2}$	$6.92 \cdot 10^{-2}$	$3.83 \cdot 10^{-1}$	$5.18 \cdot 10^{-2}$
500	$4.98 \cdot 10^{-3}$	$1.05 \cdot 10^{-2}$	$1.20 \cdot 10^{-1}$	$9.34 \cdot 10^{-2}$	$3.08 \cdot 10^{-1}$	$3.90 \cdot 10^{-2}$
1,000	$9.76 \cdot 10^{-3}$	$1.98 \cdot 10^{-2}$	$2.55 \cdot 10^{-1}$	$4.90 \cdot 10^{-2}$	$3.75 \cdot 10^{-1}$	$4.62 \cdot 10^{-2}$
2,000	$2.04 \cdot 10^{-2}$	$4.48 \cdot 10^{-2}$	$5.47 \cdot 10^{-1}$	$1.92 \cdot 10^{-2}$	$2.27 \cdot 10^{-1}$	$1.86 \cdot 10^{-2}$
5,000	$5.27 \cdot 10^{-2}$	$1.07 \cdot 10^{-1}$	1.48	$3.22 \cdot 10^{-2}$	$3.40 \cdot 10^{-1}$	$2.98 \cdot 10^{-2}$
10,000	$1.13 \cdot 10^{-1}$	$2.25 \cdot 10^{-1}$	3.11	$8.73 \cdot 10^{-2}$	$3.59 \cdot 10^{-1}$	$3.06 \cdot 10^{-2}$
20,000	$2.23 \cdot 10^{-1}$	$4.61 \cdot 10^{-1}$	6.56	$3.06 \cdot 10^{-2}$	$2.48 \cdot 10^{-1}$	$1.80 \cdot 10^{-2}$
50,000	$5.76 \cdot 10^{-1}$	1.35	$1.77 \cdot 10^{01}$	$3.02 \cdot 10^{-2}$	$3.44 \cdot 10^{-1}$	$2.35 \cdot 10^{-2}$
100,000	1.18	2.88	$3.77 \cdot 10^{01}$	$1.99 \cdot 10^{-2}$	$2.52 \cdot 10^{-1}$	$1.76 \cdot 10^{-2}$
200,000	2.49	5.21	$7.92 \cdot 10^{01}$	$4.97 \cdot 10^{-2}$	$2.66 \cdot 10^{-1}$	$2.20 \cdot 10^{-2}$
500,000	6.66	$1.37 \cdot 10^{01}$	$2.11 \cdot 10^{02}$	$7.78 \cdot 10^{-2}$	$1.70 \cdot 10^{-1}$	$1.26 \cdot 10^{-2}$
1,000,000	$1.42 \cdot 10^{01}$	$3.35 \cdot 10^{01}$	$4.49 \cdot 10^{02}$	$3.23 \cdot 10^{-2}$	$3.01 \cdot 10^{-1}$	$1.50 \cdot 10^{-2}$
2,000,000	$2.94 \cdot 10^{01}$	$5.11 \cdot 10^{01}$	$9.34 \cdot 10^{02}$	$7.04 \cdot 10^{-2}$	$3.16 \cdot 10^{-1}$	$1.26 \cdot 10^{-2}$
5,000,000	$9.26 \cdot 10^{01}$	$1.96 \cdot 10^{02}$	$2.54 \cdot 10^{03}$	$8.01 \cdot 10^{-2}$	$1.91 \cdot 10^{-1}$	$1.58 \cdot 10^{-2}$
10,000,000	$1.80 \cdot 10^{02}$	$3.48 \cdot 10^{02}$	$5.27 \cdot 10^{03}$	$1.69 \cdot 10^{-1}$	$2.87 \cdot 10^{-1}$	$1.44 \cdot 10^{-2}$

being completely empty or full is equivalent to a nested constraint of BATTERY being tight. When the charging rates of the battery are large, the battery is better able to, at a given moment, flatten large peaks or drops in power consumption. However, the latter is also dependent on whether there is enough space (energy) left in the battery to store (dispatch) this energy, which is more likely when the battery capacity is large. Thus, when adopting a large battery for load profile flattening, it is less likely that it will be completely empty or full.

Although the ratio between the execution times of ALG_{seq} and ALG_{inf} appears to depend significantly on the battery size, the maximum and CoV of the execution times of ALG_{seq} is on average around 1.9 and 3.0 times smaller than that of ALG_{inf} respectively. This means that the execution times of ALG_{seq} are on average more stable than those of ALG_{inf}, regardless of the battery size. For DEM in general and DEMKit in particular, this is beneficial since the coordination and optimization of schedules for different devices is often done in parallel due to the decentralized nature of the coordination (see, e.g., Hoogsteen et al. (2018)). As a consequence, the execution time of the entire coordination and optimization framework is constrained by the maximum execution time required for solving one (subset of) device-level optimization problem(s). Thus, using ALG_{seq} instead of ALG_{inf} within such a framework may significantly reduce the overall execution time of the framework.

In the following, we present and discuss the results of the scalability evaluation. Fig. 4 shows the execution times of the three algorithms ALG_{seq}, ALG_{inf}, and ALG_{dec}, and Table 11 shows for each studied instance size n the mean and CoV of the execution times of the corresponding instances. The added regression lines in Fig. 4 are the fitted power laws of the execution times, i.e., for each algorithm we fit the function $\phi(n) = c_1 \cdot n^{c_2}$ to the execution times. These lines indicate that the practical execution time of all three algorithms is close to $O(n)$, with ALG_{seq} and ALG_{inf} being very close together and closer to $O(n)$ than ALG_{dec}. However, the CoVs for ALG_{inf} are around one order of magnitude larger than those of both ALG_{seq} and ALG_{dec}. This suggests that the execution time of the latter two algorithms is significantly less affected by the choice of problem parameters than ALG_{inf}. This is in line the results of the comparison of the tailored implementation of ALG_{seq} for BATTERY with that of ALG_{inf} within DEMKit.

The results in Fig. 4 and Table 11 indicate that on average ALG_{seq} is 26.7 times faster than ALG_{dec} and 2.14 times faster than ALG_{inf}. With regard to the performance of ALG_{dec}, we acknowledge that ALG_{dec} and in particular the updating scheme for the single-variable bounds can probably be implemented more efficiently than in the current implementation. To reduce the influence of the overall implementation on the results of this study, we measured the total time that is spent in ALG_{dec} on solving QRAP subproblems and compared this to the execution times of ALG_{seq} and ALG_{inf}. This alternative time represents the time that is minimally required to solve all QRAP subproblems regardless of the implementation of the scheme used to update the single-variable bounds. These measurements indicate that on average around 61% of the total execution time of ALG_{dec} is spent on solving QRAP subproblems. However, this time is still on average 16.3 times more than the execution time of ALG_{seq} and 8.3 times more than that of ALG_{inf}.

Furthermore, Table 12 reports on several implementation-independent statistics of the algorithm execution. More precisely, the table shows the mean and CoV of the number of executed breakpoint operations in ALG_{seq} and the number of solved QRAP subproblems in ALG_{inf} and ALG_{dec}. These operations and subproblems are the computational bottleneck of the corresponding algorithms and thus these results provide additional insight in the practical performance of the algorithms.

Table 12 indicates that the number of breakpoint operations within ALG_{seq} appears to grow linearly in n. This is in line with the proof of Lemma 6 where we stated that this number is $O(n)$. Moreover, the number of QRAP subproblems that is solved in ALG_{inf} is significantly less than in ALG_{dec}. This is in line with the findings in Wu et al. (2021), who studied among others QRAP-NC with discrete decision variables. One reason for this is that the decomposition within ALG_{dec} algorithm always happens through the middle, i.e., a QRAP subproblem in k variables is split into two smaller QRAP subproblems in $\lfloor \frac{k}{2} \rfloor$ and $\lceil \frac{k}{2} \rceil$ respectively. As a consequence, the number of solved QRAP subproblems for an instance with n variables is always given by $8n - 4$ (see also Vidal et al. (2019) and Wu et al. (2021)). This also explains why the CoV for ALG_{dec} is always zero.

6. Conclusions

We proposed an $O(n \log n)$ time algorithm for quadratic resource allocation problems with lower and upper bound constraints on nested sums of variables. As opposed to existing algorithms with the same time complexity, our algorithm can achieve the $O(n \log n)$ time complexity using only basic data structures and is therefore easier to implement. In computational experiments, we demonstrate the good practical performance of our approach, both on synthetic data and on instances from the application area of decentralized energy management (DEM) for smart grids that use measured power consumption data as input.

Our approach builds upon monotonicity arguments that find their origin in the validity of greedy algorithms for convex optimization problems over submodular constraints (Hochbaum, 1994; Hochbaum and Hong, 1995). Such monotonicity arguments have been primarily studied for resource allocation problems where the objective function is separable, i.e., can be written as the sum of single-variable functions. However, in previous work (Schoot Uiterkamp et al., 2020) we prove the validity of similar monotonicity arguments to solve a *nonseparable* resource allocation problem with so-called generalized bound constraints. Moreover, recent results on the use of interior-point methods for nested resource allocation problems (Slager, 2019; Wright and Lim, 2020) suggest that incorporating specific nonseparable terms in the objective function does not increase the complexity of the used solution method. Thus, one interesting direction for future research is to investigate whether one can use monotonicity arguments to derive efficient algorithms for resource allocation problems over nested constraints with nonseparable objective functions.

With regard to the application within DEM systems, we compared our algorithm with an existing implementation of the state-of-the-art algorithm of van der Klauw et al. (2017) within a simulation tool for DEM research. One of our objectives was to decide which of these two algorithm is more suitable to use for a given (type of) problem instance. It would be worthwhile to conduct a more thorough comparison and

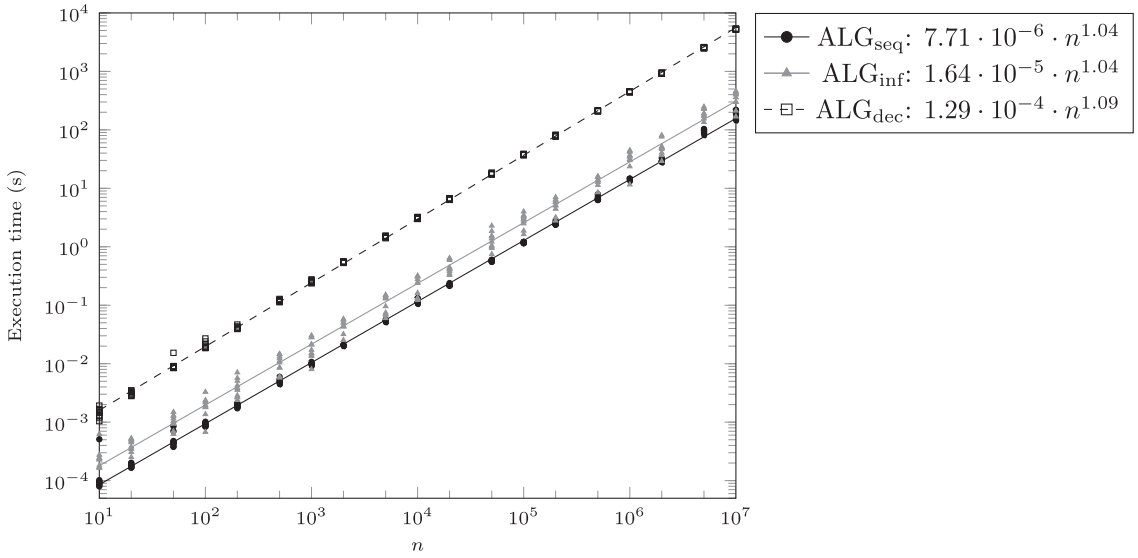


Fig. 4. Execution times of ALG_{seq} (circles, black), ALG_{inf} (triangles, gray), and ALG_{dec} (squares, open).

Table 12
Mean and coefficient of variation of the number of breakpoint operations (ALG_{seq}) and number of solved QRAP subproblems (ALG_{inf} and ALG_{dec}).

n	Mean			CoV		
	Breakpoint operations	QRAP subproblems		Breakpoint operations	QRAP subproblems	
	ALG_{seq}	ALG_{inf}	ALG_{dec}	ALG_{seq}	ALG_{inf}	ALG_{dec}
10	70.2	10.1	76	$9.70 \cdot 10^{-2}$	$4.15 \cdot 10^{-1}$	0
20	151.8	15.2	156	$4.58 \cdot 10^{-2}$	$3.37 \cdot 10^{-1}$	0
50	396.3	33.0	396	$5.64 \cdot 10^{-2}$	$4.77 \cdot 10^{-1}$	0
100	820.2	40.4	796	$3.45 \cdot 10^{-2}$	$4.93 \cdot 10^{-1}$	0
200	1,653.9	50.8	1,596	$1.74 \cdot 10^{-2}$	$6.34 \cdot 10^{-1}$	0
500	4,149.3	112.2	3,996	$1.53 \cdot 10^{-2}$	$5.71 \cdot 10^{-1}$	0
1,000	8,315.1	154.2	7,996	$1.10 \cdot 10^{-2}$	$8.16 \cdot 10^{-1}$	0
2,000	16,655.1	258.6	15,996	$6.99 \cdot 10^{-3}$	$5.91 \cdot 10^{-1}$	0
5,000	41,744.7	347.4	39,996	$4.23 \cdot 10^{-3}$	$8.08 \cdot 10^{-1}$	0
10,000	83,438.6	564.8	79,996	$2.28 \cdot 10^{-3}$	$7.95 \cdot 10^{-1}$	0
20,000	166,985.2	610.2	159,996	$2.61 \cdot 10^{-3}$	$7.94 \cdot 10^{-1}$	0
50,000	417,593.7	1,586.4	399,996	$2.42 \cdot 10^{-3}$	$8.12 \cdot 10^{-1}$	0
100,000	835 373.5	2,273.4	799,996	$6.72 \cdot 10^{-4}$	$6.06 \cdot 10^{-1}$	0
200,000	1,671,822.2	2,304.0	1,599,996	$7.61 \cdot 10^{-4}$	$7.01 \cdot 10^{-1}$	0
500,000	4,179,134.5	3,375.4	3,999,996	$4.88 \cdot 10^{-4}$	$7.52 \cdot 10^{-1}$	0
1,000,000	8,360,418.0	8,048.6	7,999,996	$1.98 \cdot 10^{-4}$	$5.99 \cdot 10^{-1}$	0
2,000,000	16,720,079.0	5,234.0	15,999,996	$1.73 \cdot 10^{-4}$	1.15	0
5,000,000	41,799,020.3	18,175.2	39,999,996	$1.13 \cdot 10^{-4}$	$6.82 \cdot 10^{-1}$	0
10,000,000	83,608,186.8	18,871.4	79,999,996	$9.08 \cdot 10^{-5}$	$8.64 \cdot 10^{-1}$	0

to develop an automated procedure to decide which algorithm is most likely to be faster. Moreover, the nonseparable version of the studied problem mentioned in the previous paragraph is related to energy management of batteries in three-phase distribution networks, where load profile flattening on all three phases together is required to avoid blackouts in these networks (Weckx and Driesen, 2015; Hoogsteen et al., 2017; Schoot Uiterkamp et al., 2020). Thus, research in this direction is also relevant in the context of DEM.

CRediT authorship contribution statement

Martijn H.H. Schoot Uiterkamp: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft, Writing - review & editing. **Johann L. Hurink:** Conceptualization, Methodology, Validation, Writing - review & editing, Supervision. **Marco E.T. Gerards:** Conceptualization, Validation, Writing - review & editing, Supervision.

Acknowledgments

The authors thank the anonymous referees for their thorough assessment of the manuscript and their constructive feedback. This research has been conducted within the SIMPS project (647.002.003) supported by NWO, The Netherlands and Eneco, The Netherlands.

Appendix. Proofs of Lemmas 1, 4, 5, 7, and 8

A.1. Proof of Lemma 1

Lemma 1. *If $L^j \le A \le B \le U^j$, we have $\mathbf{x}^j(A) \le \mathbf{x}^j(B)$ for a given $j \in \mathcal{N}$.*

Proof. For convenience, we include the equality constraint (4) into the nested constraints (5) by replacing these nested constraints by

$$\tilde{L}^k \le \sum_{i \in \mathcal{N}^k} x_i \le \tilde{U}^k, \quad k \in \mathcal{N}^j,$$

where $\tilde{L}^k = L^k$ and $\tilde{U}^k = U^k$ for $k < j$, and $\tilde{L}^j = \tilde{U}^j = C$. The Karush–Kuhn–Tucker (KKT) optimality conditions (see, e.g., Boyd and Vandenberghe (2004)) for the subproblem QRAP-NC$^j(C)$ are as follows:

$$\frac{x_i}{a_i} + \sum_{k=i}^{j}(\eta_k^j - \zeta_k^j) + \mu_i^j - \nu_i^j = 0, \quad i \in \mathcal{N}^j, \tag{A.1a}$$

$$\tilde{L}^i \le \sum_{k\in\mathcal{N}^i} x_k \le \tilde{U}^i, \quad i \in \mathcal{N}^j, \tag{A.1b}$$

$$\eta_i^j\left(\tilde{U}^i - \sum_{k\in\mathcal{N}^i} x_k\right) = 0, \quad i \in \mathcal{N}^j, \tag{A.1c}$$

$$\zeta_i^j\left(\sum_{k\in\mathcal{N}^i} x_k - \tilde{L}^i\right) = 0, \quad i \in \mathcal{N}^j, \tag{A.1d}$$

$$\mu_i^j(u_i - x_i) = 0, \quad i \in \mathcal{N}^j, \tag{A.1e}$$

$$\nu_i^j(x_i - l_i) = 0, \quad i \in \mathcal{N}^j, \tag{A.1f}$$

$$\eta_i^j, \zeta_i^j, \mu_i^j, \nu_i^j \ge 0, \quad i \in \mathcal{N}^j. \tag{A.1g}$$

Let $(\boldsymbol{\zeta}^j(C), \boldsymbol{\eta}^j(C), \boldsymbol{\mu}^j(C), \boldsymbol{\nu}^j(C))$ denote the Lagrange multipliers corresponding to the optimal solution $\boldsymbol{x}^j(C)$. Thus, $(\boldsymbol{x}^j(C), \boldsymbol{\zeta}^j(C), \boldsymbol{\eta}^j(C), \boldsymbol{\mu}^j(C), \boldsymbol{\nu}^j(C))$ satisfy the KKT-conditions (A.1).

Suppose that there exists an index $s \in \mathcal{N}$ such that $x_s^j(A) > x_s^j(B)$. Let r be the largest index with $r \le s$ such that $\sum_{k\in\mathcal{N}^{r-1}} x_k^j(A) \ge \sum_{k\in\mathcal{N}^{r-1}} x_k^j(B)$, and let t be the smallest index with $t \ge s$ such that $\sum_{k\in\mathcal{N}^t} x_k^j(A) \le \sum_{k\in\mathcal{N}^t} x_k^j(B)$. By definition of r, s, and t, we have that

$$\sum_{i=r}^{t} x_i^j(B) = \sum_{i\in\mathcal{N}^t} x_i^j(B) - \sum_{i\in\mathcal{N}^{r-1}} x_i^j(B) \ge \sum_{i\in\mathcal{N}^t} x_i^j(A) - \sum_{i\in\mathcal{N}^{r-1}} x_i^j(A) = \sum_{i=r}^{t} x_i^j(A).$$

Moreover, observe that we cannot have $r = s = t$ simultaneously. Indeed, if $r = s = t$, then we have by definition of r, s, and t that

$$\sum_{k\in\mathcal{N}^s} x_k^j(A) \le \sum_{k\in\mathcal{N}^s} x_k^j(B) \le \sum_{k\in\mathcal{N}^{s-1}} x_k^j(A) + x_s^j(B).$$

This implies $x_s^j(A) \le x_s^j(B)$, which is a contradiction. Thus, either $r < s$ or $s < t$ or both.

We show that we obtain a contradiction if $r < s$. The proof for the case where $s < t$ is symmetrical. If $r < s$, the following holds:

- By definition of r and s, we have

$$\sum_{k\in\mathcal{N}^r} x_k^j(A) < \sum_{k\in\mathcal{N}^r} x_k^j(B) = \sum_{k\in\mathcal{N}^{r-1}} x_k^j(B) + x_r^j(B) \le \sum_{k\in\mathcal{N}^{r-1}} x_k^j(A) + x_r^j(B).$$

  Thus, $x_r^j(A) < x_r^j(B)$.
- For each k such that $r \le k \le s-1$, we have by definition of r and s and KKT-condition (A.1b) that

$$\tilde{L}^k \le \sum_{i\in\mathcal{N}^k} x_i^j(A) < \sum_{i\in\mathcal{N}^k} x_i^j(B) \le \tilde{U}^k.$$

  It follows from KKT-conditions (A.1c), (A.1d), and (A.1g) that $\zeta_k^j(B) = \eta_k^j(A) = 0$. Thus, for each $r \le k \le s-1$, we have

$$\sum_{i=k}^{j}(\eta_i^j(A) - \zeta_i^j(A)) - \sum_{i=k+1}^{j}(\eta_i^j(A) - \zeta_i^j(A)) = \eta_k^j(A) - \zeta_k^j(A) \le 0$$

  and

$$\sum_{i=k}^{j}(\eta_i^j(B) - \zeta_i^j(B)) - \sum_{i=k+1}^{j}(\eta_i^j(B) - \zeta_i^j(B)) = \eta_k^j(B) - \zeta_k^j(B) \ge 0.$$

  In particular, this implies that

$$\sum_{i=r}^{j}(\eta_i^j(A) - \zeta_i^j(A)) \le \sum_{i=s}^{j}(\eta_i^j(A) - \zeta_i^j(A)) \tag{A.2}$$

  and

$$\sum_{i=r}^{j}(\eta_i^j(B) - \zeta_i^j(B)) \ge \sum_{i=s}^{j}(\eta_i^j(B) - \zeta_i^j(B)). \tag{A.3}$$

- We have $l_r \le x_r^j(A) < x_r^j(B) \le u_r$. It follows from KKT-conditions (A.1e)–(A.1g) that

$$\nu_r^j(B) = \mu_r^j(A) = 0. \tag{A.4}$$

  Similarly, since $l_s \le x_s^j(B) < x_s^j(A) \le u_s$, we have by KKT-conditions (A.1e)–(A.1g) that

$$\nu_s^j(A) = \mu_s^j(B) = 0. \tag{A.5}$$

We can now derive a contradiction as follows:

$$\sum_{i=s}^{j}(\eta_i^j(A) - \zeta_i^j(A)) = -\frac{x_s^j(A)}{a_s} - \mu_s^j(A) + \nu_s^j(A) \tag{A.6a}$$

$$< -\frac{x_s^j(B)}{a_s} - \mu_s^j(B) + \nu_s^j(B) \tag{A.6b}$$

$$= \sum_{i=s}^{j}(\eta_i^j(B) - \zeta_i^j(B)) \tag{A.6c}$$

$$\le \sum_{i=r}^{j}(\eta_i^j(B) - \zeta_i^j(B)) \tag{A.6d}$$

$$= -\frac{x_r^j(B)}{a_r} - \mu_r^j(B) + \nu_r^j(B) \tag{A.6e}$$

$$< -\frac{x_r^j(A)}{a_r} - \mu_r^j(A) + \nu_r^j(A) \tag{A.6f}$$

$$= \sum_{i=r}^{j}(\eta_i^j(A) - \zeta_i^j(A)) \tag{A.6g}$$

$$\le \sum_{i=s}^{j}(\eta_i^j(A) - \zeta_i^j(A)). \tag{A.6h}$$

Here,

- (A.6a), (A.6c), (A.6e), and (A.6g) follow from KKT-condition (A.1a);
- (A.6b) follows from Eq. (A.5) and the fact that $x_s^j(A) > x_s^j(B)$ and $a_s > 0$;
- (A.6d) follows from Eq. (A.3);
- (A.6f) follows from Eq. (A.4) and the fact that $x_r^j(A) < x_r^j(B)$ and $a_s > 0$;
- (A.6h) follows from Eq. (A.2).

It follows that $x_s^j(A) \le x_s^j(B)$. □

A.2. Proof of Lemma 4

Lemma 4. *We have $\chi^n = \kappa^n = \lambda^n$. Moreover, for each $j \in \mathcal{N}^{n-1}$, we have:*

1. *$\chi^{j+1} \le \kappa^j$ implies $\sum_{i\in\mathcal{N}^j} x_i^n(R) = L^j$ and $\chi^j = \kappa^j$;*
2. *$\lambda^j \le \chi^{j+1}$ implies $\sum_{i\in\mathcal{N}^j} x_i^n(R) = U^j$ and $\chi^j = \lambda^j$,*
3. *$\kappa^j < \chi^{j+1} < \lambda^j$ implies $L^j < \sum_{i\in\mathcal{N}^j} x_i^n(R) < U^j$ and $\chi^j = \chi^{j+1}$.*

Proof. We have $\chi^n = \kappa^n = \lambda^n$ since we defined $L^n = U^n = R$ and by definition of the solution $\boldsymbol{x}^n(R)$ the nested constraints $L^n \le \sum_{i\in\mathcal{N}} x_i^n(L^n)$ and $\sum_{i\in\mathcal{N}} x_i^n(U^n) \le U^n$ are tight. We prove the lemma by considering each of its three cases separately for each $j < n$:

1. We prove this part of the lemma for the case that j is the largest index smaller than v^{j+1} such that $\chi^{j+1} \le \kappa^j$, i.e., $\chi^{k+1} > \kappa^k$ for all $k \in \{j+1, \dots, v^{j+1}-1\}$. Using this result, we show as follows that the other case, i.e., both the situations where either $j = v^{j+1}$ or where there exists an index $k > j$ that it is the largest index in the set $\{j+1, \dots, v^{j+1}-1\}$ such that $\chi^{k+1} \le \kappa^k$, leads to a

contradiction. In the former situation, it follows that $j+1 > j = v^{j+1} \geq j+1$, which is a contradiction. In the latter situation, the lemma applies for k, meaning that $\sum_{i\in\mathcal{N}^k} x_i^n(R) = L^k$ and thus $v^k = k$. However, we also have by definition of v^{j+1} that $v^k = v^{j+1}$ since $j+1 \leq k < v^{j+1}$. This implies $k = v^{j+1}$, which is a contradiction.

If $\chi^{j+1} \leq \kappa^j$, it follows from the lower breakpoint relations in Eq. (9) that we have either $\alpha_i^{j+1} \geq \kappa^j \geq \chi^{j+1}$ (if $\kappa^j < \beta_i^j$) or $\alpha_i^{j+1} = \beta_i^j \leq \kappa^j \leq \lambda^j$ (if $\beta_i^j \leq \kappa^j$) for all $i \leq j+1$. We show that in both cases it holds that $x_i^{v^{j+1}}(V^{v^{j+1}}) = x_i^j(L^j)$:

- In the former case, note that $\alpha_i^k \leq \alpha_i^{k+1}$ for all $k < n$ by Eq. (9) and that $\chi^{j+1} = \chi^k$ for all $k \in \{j+1, \dots, v^{j+1}\}$ by definition of χ^{j+1}. Since $\chi^{k+1} > \kappa^k$ for all $k \in \{j+1, \dots, v^{j+1}-1\}$, we have that $\alpha_i^k \geq \alpha_i^{j+1} \geq \chi^{j+1} = \chi^{k+1} > \kappa^k$ for all $k \in \{j+1, \dots, v^{j+1}-1\}$. Thus, $x_i^k(L^k) = \bar{l}_i^k = x_i^{k-1}(L^{k-1})$ for all $k \in \{j+1, \dots, v^{j+1}-1\}$, which implies that $x_i^j(L^j) = x_i^{v^{j+1}-1}(L^{v^{j+1}-1})$. Moreover, note that since $\alpha^{v^{j+1}} \geq \alpha^{j+1} \geq \chi^{j+1} = \chi^{v^{j+1}}$, we have that $x_i^{v^{j+1}}(V^{v^{j+1}}) = x_i^{v^{j+1}-1}(L^{v^{j+1}-1})$. It follows that $x_i^{v^{j+1}}(V^{v^{j+1}}) = x_i^j(L^j)$.
- The latter case implies that $x_i^j(L^j) = x_i^j(U^j) = \bar{u}_i^j$. It follows by Lemmas 1 and 2 that $x_i^{v^{j+1}}(V^{v^{j+1}}) \leq x_i^{v^{j+1}}(U^{v^{j+1}}) \leq x_i^j(U^j) = x_i^j(L^j) \leq x_i^{v^{j+1}}(L^{v^{j+1}}) \leq x_i^{v^{j+1}}(V^{v^{j+1}})$.

On the one hand, if $V^{v^{j+1}} = L^{v^{j+1}}$, we have

$$\begin{aligned} L^j &= \sum_{i\mathcal{N}^j} x_i^j(L^j) = \sum_{i\in\mathcal{N}^j} x_i^{v^{j+1}}(L^{v^{j+1}}) = L^{v^{j+1}} - \sum_{i=j+1}^{v^{j+1}} x_i^{v^{j+1}}(L^{v^{j+1}}) \\ &\geq \sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^n(R) - \sum_{i=j+1}^{v^{j+1}} x_i^n(R) \\ &= \sum_{i\in\mathcal{N}^j} x_i^n(R) \geq L^j, \end{aligned}$$

where the inequality follows since $\sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^n(R) = L^{v^{j+1}}$ and by Lemma 2. On the other hand, if $V^{v^{j+1}} = U^{v^{j+1}}$, we have by Lemma 2 that

$$L^j = \sum_{i\in\mathcal{N}^j} x_i^j(L^j) = \sum_{i\in\mathcal{N}^j} x_i^{v^{j+1}}(U^{v^{j+1}}) \geq \sum_{i\in\mathcal{N}^j} x_i^n(U^n) \geq L^j.$$

In both cases, it follows that $\sum_{i\in\mathcal{N}^j} x_i^n(R) = L^j$, from which it follows directly that $\chi^j = \kappa^j$.

2. The proof for the case $\lambda^j \geq \chi^{j+1}$ is analogous to the proof for the case $\chi^{j+1} \leq \kappa^j$.
3. Suppose that $x_i^j(L^j) = x_i^n(L^n)$ holds for all $i < j+1$. By Lemma 2, this implies that $x_i^k(L^k) = x_i^j(L^j) = x_i^n(L^n)$ for all $k \in \{j, \dots, n\}$ and $i < j+1$. In particular, we have that $x_i^k(L^k) = \bar{l}_i^k$ for all $k \in \{j+1, \dots, n\}$, which implies that $\kappa^k \leq \alpha_i^k$. Furthermore, note that for any $k' \in \mathcal{N}$ there is at least one index $i_{k'} \leq k$ such that $\alpha_{i_{k'}}^{k'} \leq \kappa^{k'} < \beta_{i_{k'}}^{k'}$. Otherwise, there exists $\epsilon > 0$ such that $\kappa^{k'} + \epsilon$ is an optimal Lagrange multiplier. It follows from the relation between $\alpha_{i_{k'}}^{k'}$ and $\alpha_{i_{k'}}^{k'+1}$ in Eq. (9) that $\alpha_{i_{k'}}^{k'+1} = \kappa^{k'}$ for any $k' < n$. This implies in particular that $\alpha_{i_k}^{k+1} = \kappa^k \leq \alpha_{i_{k-1}}^k$ for all $k \in \{j+1, \dots, n\}$. It follows that $\kappa^{v^{j+1}} \leq \alpha_{i_j}^{j+1} = \kappa^j$ and thus that $\kappa^{v^{j+1}} < \chi^{j+1} = \chi^{v^{j+1}}$. Since $\chi^{v^{j+1}} \in \{\kappa^{v^{j+1}}, \lambda^{v^{j+1}}\}$, we have $\chi^{v^{j+1}} = \lambda^{v^{j+1}}$, from which it follows that $\sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^n(R) = U^{v^{j+1}}$. However, this implies that

$$\begin{aligned} \sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^{v^{j+1}}(L^{v^{j+1}}) &= \sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^{v^{j+1}}(R) = U^{v^{j+1}} \geq \sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^{v^{j+1}}(U^{v^{j+1}}) \\ &\geq \sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^{v^{j+1}}(L^{v^{j+1}}). \end{aligned}$$

This implies that $\sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^{v^{j+1}}(L^{v^{j+1}}) = \sum_{i\in\mathcal{N}^{v^{j+1}}} x_i^{v^{j+1}}(U^{v^{j+1}})$, from which it follows that $L^{v^{j+1}} = U^{v^{j+1}}$ by the monotonicity of $x^{v^{j+1}}(\cdot)$ as proven in Lemma 1. However, this is a contradiction with the assumption that $L^k < U^k$ for all $k < n$. Hence, there must be at least one index i' such that $x_{i'}^j(L^j) < x_{i'}^n(R)$. It follows that $L^j = \sum_{i\in\mathcal{N}^j} x_i^j(L^j) < \sum_{i\in\mathcal{N}^j} x_i^n(R)$.

To prove that $\sum_{i\in\mathcal{N}^j} x_i^n(R) < U^j$, we can use a similar argument wherein we show that the proposition $x_i^j(U^j) = x_i^n(R)$ cannot be true for all $i < n$. Together, this implies that $L^j < \sum_{i\in\mathcal{N}^j} x_i^n(R) < U^j$, from which it follows directly that $\chi^j = \chi^{v^{j+1}} = \chi^{j+1}$. □

A.3. Proof of Lemma 5

Lemma 5. *For each $i \in \mathcal{N}$, we have*

$$x_i^n(R) = \begin{cases} l_i & \text{if } \chi^i < \alpha_i^i, \\ a_i\chi^i & \text{if } \alpha_i^i \leq \chi^i < \beta_i^i, \\ u_i & \text{if } \beta_i^i \leq \chi^i. \end{cases}$$

Proof. Let $\mathcal{J}$ denote the set of indices whose corresponding nested lower or upper constraint is tight in $\boldsymbol{x}^n(R)$. More precisely,

$$\mathcal{J} := \{v^j \mid j \in \mathcal{N}\} \equiv \{j_1, \dots, j_q\},$$

where $q := |\mathcal{J}|$ and $j_1 < \cdots < j_q$. For a given $p \in \{1, \dots, q\}$, note that since either the lower or upper nested constraint corresponding to j_p is tight in the solution $\boldsymbol{x}^n(R)$, we have that $\sum_{i\in\mathcal{N}^{j_p}} x_i^n(R) = V^{j_p}$. This implies that the vector $(x_i^n(R))_{1\leq i\leq j_p}$ is the optimal solution to the subproblem QRAP-NC$^{j_p}(V^{j_p})$, i.e., to the problem

$$\begin{aligned} \text{QRAP-NC}^{j_p}(V^{j_p}) \quad : \quad \min_{\boldsymbol{x}\in\mathbb{R}^{j_p}} \quad & \sum_{i\in\mathcal{N}^{j_p}} \frac{1}{2}\frac{x_i^2}{a_i} \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}^{j_p}} x_i = V^{j_p}, \\ & L^k \leq \sum_{i\in\mathcal{N}^k} x_i \leq U^k, \quad k \in \{1, \dots, j_p - 1\}, \\ & l_i \leq x_i \leq u_i, \quad i \in \{1, \dots, j_p\}. \end{aligned} \tag{A.7}$$

Note that in the optimal solution $(x_i^n(R))_{i\in\mathcal{N}^{j_p}}$ to this problem, none of the nested constraints (A.7) for k with $j_{p-1} < k < j_p$ are tight. As a consequence, when deriving the reformulated equivalent problem QRAP$^{j_p}(V^{j_p})$, it follows from Lemmas 2 and 3 that we may replace the single-variable bounds (7) for i with $j_{p-1} < i < j_p$ by the original variable bounds $l_i \leq x_i \leq u_i$. Thus, we can reformulate QPRAP-NC$^{j_p}(V^{j_p})$ to

$$\begin{aligned} \text{QRAP}^{j_p}(V^{j_p}) \quad : \quad \min_{\boldsymbol{x}\in\mathbb{R}^{j_p}} \quad & \sum_{i\in\mathcal{N}^{j_p}} \frac{1}{2}\frac{x_i^2}{a_i} \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}^{j_p}} x_i = V^{j_p}, \\ & x_i^{j_p-1}(L^{j_p-1}) \leq x_i \leq x_i^{j_p-1}(U^{j_p-1}), \\ & \quad i \in \{1, \dots, j_{p-1}\}, \\ & l_j \leq x_j \leq u_j, \quad i \in \{j_{p-1}+1, \dots, j_p\}. \end{aligned}$$

Recall that χ^{j_p} is the optimal Lagrange multiplier for this problem. As a consequence, we can directly compute $x_i^{j_p}(R)$ for $i \in \{j_{p-1}+1, \dots, j_p\}$ using Eq. (3):

$$x_i^n(R) = x_i^{j_p}(V^{j_p}) = \begin{cases} l_i & \text{if } \chi^{j_p} < \alpha_i^i, \\ a_i\chi^{j_p} & \text{if } \alpha_i^i \leq \chi^{j_p} < \beta_i^i, \\ u_i & \text{if } \beta_i^i \leq \chi^{j_p}. \end{cases}$$

The result of the lemma follows since we have $\chi^{j_p} = \chi^i$ for each $i \in \{j_{p-1}, \dots, j_p\}$ by definition of j_p and χ^i. □

A.4. Proof of Lemma 7

Lemma 7. *When applying ALG_{seq} to WC-1, first $\Omega(n)$ breakpoints are inserted in the breakpoint set $\mathcal{D}$ and subsequently $\Omega(n)$ breakpoints are removed from $\mathcal{D}$ throughout the course of the algorithm.*

Proof. First, note that for this instance we have $\alpha_i^i = i$ and $\beta_i^i = 2n+1-i$ for $i \in \mathcal{N}$ and $\kappa^1 = 1$ and $\lambda^1 = 2n$. Moreover, its optimal solution is given by $\bar{x}_1 = 2n + \frac{1}{2}$ and $\bar{x}_i = 2n + 1 - i$ for $i \in \mathcal{N}\backslash\{1\}$.

When we follow all steps of Algorithm 3 and the relevant procedures, we observe the following:

- In SolveSubproblems(2), we have $L^1 + \max(l_2, \min(\kappa^1, u_2)) = 1 + 2 = 3 = L^2$. Hence, the condition of the if-statement in Line 1 applies and thus $\kappa^2 = \kappa^1 = 1$. Analogously, we have $U^1 + \max(l_2, \min(\lambda^1, u_2)) = 2n + 2n - 1 = 4n - 1 = U^2$ and thus the condition of the if-statement in Line 17 applies and we have $\lambda^2 = \lambda^1 = 2n$. It follows from Lines 33–38 that both α_2^2 and β_2^2 are added to the breakpoint sets $\mathcal{A}$ and $\mathcal{B}$ respectively. Moreover, it follows from Lines 39–58 that no breakpoints are removed from these sets.
- Suppose that $\kappa^j = \kappa^1 = 1$ and $\lambda^j = \lambda^1 = 2n$ for some $j \geq 2$. Then $L^{j-1} + \max(l_j, \min(\kappa^{j-1}, u_j)) = \frac{1}{2}j(j-1) + j = \frac{1}{2}j(j+1) = L^j$ and $U^{j-1} + \max(l_j, \min(\lambda^{j-1}, u_j)) = 2(j-1)n + j - 1 - \frac{1}{2}j(j-1) + 2n + 1 - j = 2jn + j - \frac{1}{2}j(j+1) = U^j$. Following the same reasoning as for the case $j = 2$, it follows that both α_j^j and β_j^j are added to $\mathcal{A}$ and $\mathcal{B}$ respectively and that no breakpoints are removed from these sets.

By applying the principle of induction, we may conclude that all breakpoints α_i^i and β_i^i for $i \in \mathcal{N}^{n-1}$ are added to the breakpoint sets and that none of them have been removed from these sets by the start of iteration $j = n$.

Since $L^n > \frac{1}{2}n(n-1)$ and $U^n < 2n^2 + n - \frac{1}{2}n(n+1)$, the else-statements of Lines 14–15 and 30–31 apply. Thus, the values $\kappa^{n-1} \equiv \kappa^1$ and $\lambda^{n-1} \equiv \lambda^1$ are removed from the multiplier sets $\mathcal{K}$ and $\mathcal{L}$ respectively (leaving them empty), the breakpoints α_n^n and β_n^n are added to $\mathcal{A}$ and $\mathcal{B}$ respectively, and the procedures LowerSubproblem(n) and UpperSubproblem(n) are applied since $\kappa^n > \kappa^{n-1}$ and $\lambda^n < \lambda^{n-1}$. Since in the optimal solution the variable $\bar{x}_1$ is strictly in between its bounds and all other variables equal their upper bound, we know that $\beta_i^i < \kappa^n < \beta_1^1$ for all $i > 1$. Thus, in the procedures LowerSubproblem(j), all breakpoints except β_1^1 have been considered as candidate multipliers and have afterwards been removed from the corresponding breakpoint set. Summarizing, we conclude that first $\Omega(n)$ breakpoints have been inserted and subsequently $\Omega(n)$ breakpoints have been deleted. □

A.5. Proof of Lemma 8

Lemma 8. *For the instance WC-2, the Lagrange multipliers κ^j and λ^j as computed by ALG_{seq} are given for each $j \in \mathcal{N}^{n-1}$ by*

$$\kappa^j := (-1)^j \left(2j - \frac{3}{2}\right) - \frac{1}{2}, \qquad \lambda^j := (-1)^j \left(2j - \frac{3}{2}\right) + \frac{1}{2}.$$

Proof. We prove the lemma by induction. First, it follows from Lines 3–10 of Algorithm 3 that $\kappa^1 = \alpha_1^1 = l_1 = L^1 = -1$ and $\lambda^1 = \beta_1^1 = u_1 = U^1 = 0$. Also, we have $(-1)^1 \left(2 \cdot 1 - \frac{3}{2}\right) - \frac{1}{2} = -1$ and $(-1)^1 \left(2 \cdot 1 - \frac{3}{2}\right) + \frac{1}{2} = 0$, so the lemma holds for $j = 1$.

Second, suppose that the lemma holds for all $j' < j$ for some $j \in \mathcal{N}^{n-1}$. We show that this implies that the lemma also holds for j. Here, we prove the lemma for the case that j is even (the proof for the case that j is uneven is symmetrical). If j is even, then $j-1$ is uneven and thus $\lambda^{j-1} = -\left[2(j-1) - \frac{3}{2}\right] + \frac{1}{2} = -2j + 4$. This implies that

$$\begin{aligned} U^{j-1} + \max(l_j, \min(\lambda^{j-1}, u_j)) &= -(j-1) + 1 + \max(-2n, \min(-2j+4, 2n)) \\ &= -3j + 6 < j + 1 = U^j. \end{aligned}$$

Thus, the condition in the if-statement of Line 26 of Algorithm 4 is satisfied and we have

$$\lambda^j = U^j - U^{j-1} = j + 1 - [-(j-1) + 1] = 2j - 1 = (-1)^j \left(2j - \frac{3}{2}\right) + \frac{1}{2}.$$

Furthermore, note that $\lambda^{j-1} < \kappa^j$ since otherwise

$$\begin{aligned} L^j &= \sum_{i \in \mathcal{N}^j} x_i^j(L^j) = \sum_{i \in \mathcal{N}^{j-1}} x_i^j(L^j) + \max(l_i, \min(\kappa^j, u_i)) \\ &\leq U^{j-1} + \max(l_i, \min(\lambda^{j-1}, u_i)) = -3j + 6 < j = L^j, \end{aligned}$$

which is a contradiction. This implies that $x_i^{j-1}(U^{j-1}) \leq x_i^j(L^j)$ for all $i \leq j-1$ and thus that

$$L^j = \sum_{i \in \mathcal{N}^j} x_i^j(L^j) = \sum_{i \in \mathcal{N}^{j-1}} x_i^j(L^j) + \kappa^j \geq \sum_{i \in \mathcal{N}^{j-1}} x_i^{j-1}(U^{j-1}) + \kappa^j = U^{j-1} + \kappa^j.$$

It follows that $\kappa^j = L^j - U^{j-1} = j - [-(j-1) + 1] = 2j - 2 = (-1)^j \left(2j - \frac{3}{2}\right) - \frac{1}{2}$. By the induction principle, we have thereby proven the lemma for all $j \in \mathcal{N}^{n-1}$. □

References

Akhil, P.T., Sundaresan, R., 2018. Algorithms for separable convex optimization with linear ascending constraints. Sādhanā 43 (9), 146. http://dx.doi.org/10.1007/s12046-018-0890-2.

Alexandrescu, A., 2017. Fast deterministic selection. In: Iliopoulos, C.S., Pissis, S.P., Puglisi, S.J., Raman, R. (Eds.), Leibniz International Proceedings in Informatics, LIPIcs, Vol. 75. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 24:1–24:9. http://dx.doi.org/10.4230/LIPIcs.SEA.2017.24, arXiv:1606.00484.

Arvind, A., Rangan, C., 1999. Symmetric min-max heap: A simpler data structure for double-ended priority queue. Inf. Process. Lett. 69 (4), 197–199. http://dx.doi.org/10.1016/S0020-0190(99)00014-9.

Atkinson, M.D., Sack, J.-R., Santoro, N., Strothotte, T., 1986. Min-max heaps and generalized priority queues. Commun. ACM 29 (10), 996–1000. http://dx.doi.org/10.1145/6617.6621.

Beaudin, M., Zareipour, H., 2015. Home energy management systems: A review of modelling and complexity. Renew. Sustain. Energy Rev. 45, 318–335. http://dx.doi.org/10.1016/j.rser.2015.01.046.

Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E., 1973. Time bounds for selection. J. Comput. System Sci. 7 (4), 448–461. http://dx.doi.org/10.1016/S0022-0000(73)80033-9.

Boyd, S., Vandenberghe, L., 2004. Convex Optimization, seventh ed. Cambridge University Press, Cambridge, MA.

Brass, P., 2008. Advanced Data Structures, first ed. Cambridge University Press, Cambridge, MA, http://dx.doi.org/10.1017/CBO9780511800191.

Brodal, G.S., 2013. A survey on priority queues. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (Eds.), Space-Efficient Data Structures, Streams, and Algorithms. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 150–163. http://dx.doi.org/10.1007/978-3-642-40273-9_11.

Brucker, P., 1984. An $O(n)$ algorithm for quadratic knapsack problems. Oper. Res. Lett. 3 (3), 163–166. http://dx.doi.org/10.1016/0167-6377(84)90010-5.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2009. Introduction to Algorithms, third ed. The MIT Press, Cambridge, MA.

Esther, B.P., Kumar, K.S., 2016. A survey on residential demand side management architecture, approaches, optimization models and methods. Renew. Sustain. Energy Rev. 59, 342–351. http://dx.doi.org/10.1016/j.rser.2015.12.282.

Gabow, H.N., Tarjan, R.E., 1983. A linear-time algorithm for a special case of disjoint set union. In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC). In: STOC '83, ACM, Boston, MA, pp. 246–251. http://dx.doi.org/10.1145/800061.808753.

Gabow, H.N., Tarjan, R.E., 1985. A linear-time algorithm for a special case of disjoint set union. J. Comput. System Sci. 30 (2), 209–221. http://dx.doi.org/10.1016/0022-0000(85)90014-5.

Galil, Z., Italiano, G.F., 1991. Data structures and algorithms for disjoint set union problems. ACM Comput. Surv. (CSUR) 23 (3), 319–344. http://dx.doi.org/10.1145/116873.116878.

Gerards, M.E.T., 2014. Algorithmic Power Management: Energy Minimisation under Real-Time Constraints (Ph.D. thesis). University of Twente, http://dx.doi.org/10.3990/1.9789036536790.

Gerards, M.E.T., Toersche, H.A., Hoogsteen, G., van der Klauw, T., Hurink, J.L., Smit, G.J.M., 2015. Demand side management using profile steering. In: 2015 IEEE Eindhoven PowerTech. IEEE, Eindhoven, http://dx.doi.org/10.1109/PTC.2015.7232328.

Hochbaum, D.S., 1994. Lower and upper bounds for the allocation problem and other nonlinear optimization problems. Math. Oper. Res. 19 (2), 390–409. http://dx.doi.org/10.1287/moor.19.2.390.

Hochbaum, D.S., Hong, S.-P., 1995. About strongly polynomial time algorithms for quadratic optimization over submodular constraints. Math. Program. 69, 269–309. http://dx.doi.org/10.1007/BF01585561.

Hoogsteen, G., Gerards, M.E.T., Hurink, J.L., 2018. On the scalability of decentralized energy management using profile steering. In: 2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe). IEEE, Sarajevo, http://dx.doi.org/10.1109/ISGTEurope.2018.8571823.

Hoogsteen, G., Hurink, J.L., Smit, G.J.M., 2019. Demkit: a decentralized energy management simulation and demonstration toolkit. In: 2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe). IEEE, Bucharest, http://dx.doi.org/10.1109/ISGTEurope.2019.8905439.

Hoogsteen, G., Molderink, A., Hurink, J.L., Smit, G.J., Kootstra, B., Schuring, F., 2017. Charging electric vehicles, baking pizzas, and melting a fuse in lochem. CIRED - Open Access Proc. J. 2017 (1), 1629–1633. http://dx.doi.org/10.1049/oap-cired.2017.0340.

Huang, W., Wang, Y., 2009. An optimal speed control scheme supported by media servers for low-power multimedia applications. Multimedia Syst. 15 (2), 113–124. http://dx.doi.org/10.1007/s00530-009-0153-5.

Hvattum, L.M., Norstad, I., Fagerholt, K., Laporte, G., 2013. Analysis of an exact algorithm for the vessel speed optimization problem. Netw. 62 (2), 132–135. http://dx.doi.org/10.1002/net.21503.

Ibaraki, T., Katoh, N., 1988. Resource Allocation Problems: Algorithmic Approaches, first ed. The MIT Press, Cambridge, MA.

Kiwiel, K.C., 2005. On Floyd and Rivest's SELECT algorithm. Theoret. Comput. Sci. 347 (1), 214–238. http://dx.doi.org/10.1016/j.tcs.2005.06.032.

Kiwiel, K.C., 2008. Breakpoint searching algorithms for the continuous quadratic knapsack problem. Math. Program. 112 (2), 473–491. http://dx.doi.org/10.1007/s10107-006-0050-z.

van der Klauw, T., Gerards, M.E.T., Hurink, J.L., 2017. Resource allocation problems in decentralized energy management. OR Spectr. 39 (3), 749–773. http://dx.doi.org/10.1016/0167-6377(84)90010-5.

Knuth, D.E., 1998. The Art of Computer Programming - Volume 3: Sorting and Searching, second ed. Addison-Wesley, Reading, MA.

Lund, H., Østergaard, P.A., Connolly, D., Ridjan, I., Mathiesen, B.V., Hvelplund, F., Thellufsen, J.Z., Sorknæs, P., 2016. Energy storage and smart energy systems. Int. J. Sustain. Energy Plan. Manag. 11, 3–14. http://dx.doi.org/10.5278/ijsepm.2016.11.2.

Müller-Hannemann, M., Schirra, S. (Eds.), 2010. Algorithm Engineering: Bridging the Gap between Algorithm Theory and Practice, first ed. Springer Berlin Heidelberg, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-14866-8_1.

Nagano, K., Aihara, K., 2012. Equivalence of convex minimization problems over base polytopes. Jpn. J. Ind. Appl. Math. 29 (3), 519–534. http://dx.doi.org/10.1007/s13160-012-0083-z.

Norstad, I., Fagerholt, K., Laporte, G., 2011. Tramp ship routing and scheduling with speed optimization. Transp. Res. C 19 (5), 853–865. http://dx.doi.org/10.1016/j.trc.2010.05.001.

Patriksson, M., 2008. A survey on the continuous nonlinear resource allocation problem. European J. Oper. Res. 185 (1), 1–46. http://dx.doi.org/10.1016/j.ejor.2006.12.006.

Patriksson, M., Strömberg, C., 2015. Algorithms for the continuous nonlinear resource allocation problem - new implementations and numerical studies. European J. Oper. Res. 243 (3), 703–722. http://dx.doi.org/10.1016/j.ejor.2015.01.029.

Patwary, M.M.A., Blair, J., Manne, F., 2010. Experiments on union-find algorithms for the disjoint-set data structure. In: Festa, P. (Ed.), International Symposium on Experimental Algorithms (SEA). Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 411–423. http://dx.doi.org/10.1007/978-3-642-13193-6_35.

Reijnders, V.M.J.J., Gerards, M.E.T., Hurink, J.L., Smit, G.J.M., 2018. Testing grid-based electricity prices and batteries in a field test. In: CIRED 2018 Workshop Proceedings. CIRED, Ljubljana, http://dx.doi.org/10.34890/219.

Roberts, B.P., Sandberg, C., 2011. The role of energy storage in development of smart grids. Proc. IEEE 99 (6), 1139–1144. http://dx.doi.org/10.1109/JPROC.2011.2116752.

Schoot Uiterkamp, M.H.H., Gerards, M.E.T., Hurink, J.L., Quadratic Nonseparable Resource Allocation Problems with Generalized Bound Constraints, University of Twente, ArXiv: arXiv:2007.06280. 2020.

Schoot Uiterkamp, M.H.H., Gerards, M.E.T., Hurink, J.L., 2021. On a reduction for a class of resource allocation problems. INFORMS J. Comput. in press.

Siano, P., 2014. Demand response and smart grids - a survey. Renew. Sustain. Energy Rev. 30, 461–478. http://dx.doi.org/10.1016/j.rser.2013.10.022.

Slager, J., 2019. Nonlinear Convex Optimisation Problems in the Smart Grid (B.Sc. thesis). University of Twente.

Vidal, T., Gribel, D., Jaillet, P., 2019. Separable convex optimization with nested lower and upper constraints. INFORMS J. Optim. 1 (1), 71–90. http://dx.doi.org/10.1287/ijoo.2018.0004.

Vidal, T., Jaillet, P., Maculan, N., 2016. A decomposition algorithm for nested resource allocation problems. SIAM J. Optim. 26 (2), 1322–1340. http://dx.doi.org/10.1137/140965119.

Weckx, S., Driesen, J., 2015. Load balancing with EV chargers and PV inverters in unbalanced distribution grids. IEEE Trans. Sustain. Energy 6 (2), 635–643. http://dx.doi.org/10.1109/TSTE.2015.2402834.

Wright, S.E., Lim, S., 2020. Solving nested-constraint resource allocation problems with an interior point method. Oper. Res. Lett. 48 (3), 297–303. http://dx.doi.org/10.1016/j.orl.2020.04.001.

Wu, Z., Nip, K., He, Q., 2021. A new combinatorial algorithm for separable convex resource allocation with nested bound constraints. INFORMS J. Comput. http://dx.doi.org/10.1287/ijoc.2020.1006.

Zame, K.K., Brehm, C.A., Nitica, A.T., Richard, C.L., Schweitzer III, G.D., 2018. Smart grid and energy storage: Policy recommendations. Renew. Sustain. Energy Rev. 82, 1646–1654. http://dx.doi.org/10.1016/j.rser.2017.07.011.