Continuous Optimization

Parallel subgradient algorithm with block dual decomposition for large-scale optimization

Yuchen Zheng a, Yujia Xie a, Ilbin Lee b, Amin Dehghanian b, Nicoleta Serban a,∗

a H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, 755 Ferst Dr. NW, Atlanta, GA 30332, USA
b Alberta School of Business, University of Alberta, 2-29B Business Building, Edmonton, Alberta T6G 2R6, Canada

A R T I C L E   I N F O

Article history:
Received 20 May 2021
Accepted 29 November 2021
Available online 4 December 2021

Keywords:
Distributed decision making
Community detection
Block dual decomposition
Large scale optimization
Parallel subgradient algorithm

A B S T R A C T

This paper studies computational approaches for solving large-scale optimization problems using a Lagrangian dual reformulation, solved by parallel sub-gradient methods. Since there are many possible reformulations for a given problem, an important question is: Which reformulation leads to the fastest solution time? One approach is to detect a block diagonal structure in the constraint matrix, and reformulate the problem by dualizing the constraints outside of the blocks; the approach is defined herein as block dual decomposition. Main advantage of such a reformulation is that the Lagrangian relaxation has a block diagonal constraint matrix, thus decomposable into smaller sub-problems that can be solved in parallel. We show that the block decomposition can critically affect convergence rate of the sub-gradient method. We propose various decomposition methods that use domain knowledge or apply algorithms using knowledge about the structure in the constraint matrix or the dependence in the decision variables, towards reducing the computational effort to solve large-scale optimization problems. In particular, we introduce a block decomposition approach that reduces the number of dualized constraints by utilizing a community detection algorithm. We present empirical experiments on an extensive set of problem instances including a real application. We illustrate that if the number of the dualized constraints in the decomposition increases, the computational effort within each iteration of the sub-gradient method decreases while the number of iterations required for convergence increases. The key message is that it is crucial to employ prior knowledge about the structure of the problem when solving large scale optimization problems using dual decomposition.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

Solving large-scale optimization problems can be computationally challenging due to the data storage and retrieval, and due to the computational load and memory usage for obtaining an optimal solution. Distributed and parallel computing are a popular framework for tackling the computational complexity of large-scale optimization (Androulakis, Visweswaran, & Floudas, 1996; Boyd, Parikh, Chu, Peleato, & Eckstein, 2011; Camponogara & De Oliveira, 2009; Inanhan, Stipanovic, & Tomlin, 2002; Nedic & Ozdaglar, 2009b; Palomar & Mung, 2006; Raffard, Tomlin, & Boyd, 2004; Richtárik & Takáč, 2016; Shastri, Hansen, Rodríguez, & Ting, 2011; Simonetto & Jamali-Rad, 2016; Terelius, Topcu, & Murray, 2011; Xiao, Johansson, & Boyd, 2004). Parallel computing for optimization problems involves three computational considerations: the decomposition into smaller sub-problems in a way that each sub-problem can be stored and solved independently, solving the sub-problems iteratively, and constructing a solution to the original problem from the smaller sub-problems in the decomposition (Boyd et al., 2011; Camponogara & De Oliveira, 2009; Nedic & Ozdaglar, 2009b; Palomar & Mung, 2006; Raffard et al., 2004; Shastri et al., 2011; Simonetto & Jamali-Rad, 2016; Terelius et al., 2011; Xiao et al., 2004).

1 The authors have consulted with other researchers about the terminology 'distributed optimization', 'distributed computing', 'parallel optimization', and 'parallel computing'. We found that the scope of the terms varied. In particular, there are researchers who use the term 'distributed' for the case where neither different machines can communicate with each other nor there exists a central coordinator, and the machines have their own objective. On the other hand, there are others who use the term for the case where different machines can communicate with each other but there is no central coordinator. To avoid any confusion, we use 'parallel' instead of 'distributed', but the methodology introduced in this paper may be applicable to a broader context.

https://doi.org/10.1016/j.ejor.2021.11.054
0377-2217/© 2021 Elsevier B.V. All rights reserved.
2006; Xiao et al., 2004), aircraft coordination (Inalhan et al., 2002; Raffard et al., 2004), and estimation problem in sensor networks (Duchi, Agarwal, & Wainwright, 2012).

Though many optimization problems are not initially amenable to parallel computing, they could be solved using such an approach after their reformulation. The most widely used and general reformulation frameworks are the decomposition-based approaches, which consist of Lagrangian dual, Dantzig–Wolfe decomposition, and Benders decomposition (Martin, 1999). These methods are both reformulation techniques and solution methods that can benefit from special structures of the given optimization problem. Benders’ approach is considered as the dual of Dantzig–Wolfe, which is closely related to Lagrangian dual. For simplicity of exposition, we present our results in this paper in terms of the Lagrangian dual for which we provide a more formal description as follows.

Consider the following linear program (LP):

\[
\begin{align*}
\max & \quad c^T x \\
\text{s.t.} & \quad Ax \leq d, \\
& \quad x \geq 0,
\end{align*}
\]

where \( c \) is an \( n \)-dimensional row vector, \( x \) and \( d \) are, respectively, \( n \)-dimensional and \( m \)-dimensional column vectors, and \( A \) is an \( m \times n \) matrix. Suppose that the matrix \( A \) has the bordered block-diagonal (BBD) structure, i.e., \( A \) is representable in the following form, perhaps after permutations of rows and columns:

\[
\begin{bmatrix}
D_1 & \cdots & D_k \\
B_1 & \cdots & B_k
\end{bmatrix},
\]

where the submatrix \([B_1, B_2, \ldots, B_k]\) is called the border component, and the remaining submatrix consisting of \( D_1, D_2, \ldots, D_k \) on the diagonal is called the block-diagonal component. To solve the LP by the Lagrangian approach, we may move the constraints associated with the border component to the objective function, specifying the Lagrangian relaxation, and subsequently, the Lagrangian dual (Martin, 1999). Note that the resulting Lagrangian relaxation is separable into \( k \) smaller sub-problems whose decision variables and constraints are uniquely identified by the blocks \( D_1, D_2, \ldots, D_k \), having two major advantages: (1) the smaller the sub-problems are, the easier to solve; and (2) these \( k \) smaller sub-problems may be solved in parallel. Since the Lagrangian dual problem is usually solved by sub-gradient algorithms, which require iterative solving of the Lagrangian relaxation, these two advantages can greatly improve the computational efficiency of the solution method in large-scale optimization problems (Martin, 1999).

Generally, a problem may have an unknown BBD structure, or/and it may have multiple BBD structures. Thus, the primary question is: How to detect the unknown BBD structure, or select among the multiple possible BBD structures? To detect a BBD structure, we need to identify the blocks \( D_1, D_2, \ldots, D_k \) within the constraint matrix. Since each block may uniquely be identified by the variables associated with the block, the problem of identifying the blocks may equivalently be viewed as partitioning of the variables. Hence, we use the term \textit{variable partitioning} to refer to the problem of detecting BBD structures.

On the theoretical side, the solution methods, i.e., sub-gradient algorithms, converge to optimality under some mild conditions, regardless of which constraints are dualized or how decision variables are partitioned (Bertsekas, 1995). On the practical side, this convergence may be slow, and it could be critically affected by the partition of decision variables.

In terms of practical implementation, the simplest decomposition approach is when the partition of the decision variables is fully specified using prior knowledge regarding the problem (see, e.g., Holmberg & Yuan 2000; Rehfeldt et al. 2021). However, most commonly, no prior knowledge is available for a problem. Moreover, since the prior knowledge is generally coming from the intuition regarding the problem, hence high level of the understanding of the problem, it may not capture the dependency between different variables and/or constraints. Hence, the approach may not fully reach its computational gain if any.

The focus of this paper is to study the problem of variable partitioning with the aim of maximizing computational performance of solving the Lagrangian dual using parallel sub-gradient methods. Specifically, we seek to devise data-driven approaches that can detect structures in the constraint matrix or dependence/coupling among the decision variables. For this purpose, we first investigate how the performance of sub-gradient methods is affected by the partition of decision variables. On one extreme, if all constraints are dualized, the Lagrangian relaxation is separable with respect to each variable, which leads to trivial sub-problems, with an extremely slow convergence of sub-gradient methods. Generally, whenever a large number of constraints are dualized, the Lagrangian relaxation and the original problem are vastly different, which slows down the sub-gradient method substantially. On another extreme, if no constraint is dualized, then the Lagrangian relaxation is identical to the original problem. Therefore, a viable approach should consider the trade-off between these two aspects: minimizing the number of dualized constraints, and detecting a structure amenable to parallel optimization which may be interpreted as maximizing the number of blocks within the BBD structure, hence reducing the computational effort in large-scale optimization.

Key contributions of this paper are relevant to proposing computationally amenable variable partitioning methods that balance the number of blocks against the number of dualized constraints, for large-scale optimization problems over geographic areas such as transportation and network optimization models. Within these settings, each variable is connected to a location in the geographic or network space, which is partitioned into different areas using a certain criterion, and the variables in the same area in the space are grouped together. Note that each variable group is associated with an area in the space, that has its own attributes which may be used for further analysis. This example illustrates the use of intrinsic structural or dependence knowledge about the optimization problems. We will illustrate the variable partitioning over a geographic space, but other settings could apply similarly; for example, when the space is a social network or an electric network.

Our proposed variable partitioning methods are distinguished as follows:

- One method directly considers the geographically-defined variable groups as blocks of the variable partitioning problem while other methods further refine or redefine these initial variable groups to derive larger secondary variable groups.
- Another method uses the geographic attributes of each area as the input of an unsupervised algorithm, which further merges the initial groups of variables into larger groups.
- Towards reducing the dualized constraints, we take a network analysis approach. Specifically, we propose to construct a graph representing the relationship between the initial groups of variables and the constraints of optimization problems, and apply clustering algorithms to the graph to identify subsets of variables to be grouped together. Specifically, we use the community detection algorithm from the physics literature (Newman, 2004; 2006). The goal of community detection is to identify...
community structures within a network, in other words, to find groups of nodes in such a way that the connections within each group are dense while there is little connectivity between the groups. Roughly speaking, we use community detection to group decision variables that tend to appear in constraints together so that the number of constraints that involve variables over multiple sub-problems (thus, need to be dualized) is minimized.

For illustration purpose, we first present the proposed methods applied to transportation problems and then present a general version in Section 4.3. We provide empirical illustrations for both versions of our approach, the one tailored to transportation problems and the general one. The results highlight the importance of variable partitioning and show that using structural or dependent knowledge about the problem significantly accelerates the convergence of parallel sub-gradient methods for various problem instances.

This paper is organized as follows. We follow with a section on a literature review, pointing only to a sample of related existing research. Then, we review dual decomposition and sub-gradient methods in Section 3. In this section, we also analyze why decomposition is important for the performance of sub-gradient methods. In Section 4, we introduce several partitioning methods, including the new approach using community detection. We illustrate its empirical performance using a real application and generated instances in Section 5, and conclude in Section 6.

2. Literature review

In this section, we provide a review of related literature. Specifically, we first review the literature on variable partitioning, and then provide a review on the literature of detecting a decomposition of optimization problems, particularly, with a focus on community detection approaches.

The literature on variable partitioning methods may be classified in two categories: prior knowledge and data-driven methods. In the first class, the partitioning of the variables in the optimization formulation uses prior information regarding the problem. This is the conventional approach to variable partitioning with a vast literature, e.g., Medhi (1990), Carøe & Schultz (1999), Holmberg & Yuan (2000), Parikh & Boyd (2014), Maher (2021), just naming a few. For example, Parikh & Boyd (2014) introduced a distributed block splitting algorithm based on graph projection splitting towards solving large-scale problems in parallel. However, the same paper pointed out that, in practice, it was not obvious how to partition the variables. As another example, Medhi (1990) studied how to implement a parallel algorithm with dual decomposition, but the paper assumed that a partition of decision variables was given.

In data-driven approaches, the variable partitioning is being solved using algorithmic methods, detecting structures which are unknown in advance. These algorithmic approaches first construct a hypergraph of the constraint matrix, and then employ an algorithm to partition vertices of the hypergraph. The only input of these methods is the constraint matrix, thus the methods are general in that they can be applied for any optimization formulation. Ferris & Horn (1998) is one of the earliest papers in this line of research. To address the problem of variable partitioning, they first constructed a bipartite graph corresponding to the constraint matrix, which was used to develop an optimization formulation for the variable partitioning. They used the partition in the bundle method, which is another parallel algorithm to solve the dual formulation. Aykanat, Pinar, & Catalyurek (2004) represented the nonzero structure of a matrix by proposing bipartite graph and hypergraph models that reduce the variable partitioning problem to those of graph partitioning by vertex separator and hypergraph partitioning, respectively, which they solved by a couple of the state-of-the-art graph and hypergraph partitioning tools. Their computational experiments revealed that the proposed methods are effective both in terms of solution quality and runtime, particularly over the approach of Ferris & Horn (1998).

Bergner et al. (2015) and Khaniyev, Elhedhli, & Erenay (2018) have considered variable partitioning in mixed integer programming (MIP) with the aim of finding a partition so that the corresponding Lagrangian relaxation results in a tight bound of the optimal objective value. In particular, Khaniyev et al. (2018) introduced a heuristic method to obtain a partition that used a community detection algorithm as a subroutine. However, both papers focused on improving the Lagrangian bound for MIPs, while our paper focuses on improving the empirical performance of parallel solution algorithms for large optimization problems. In their empirical results, Khaniyev et al. (2018) reported several measures of “goodness” of a partition, with the resulting Lagrangian bound as their performance measures. To the best of our knowledge, our paper is the first to apply data-driven algorithms to find a partition of variables that leads to better performance of parallel solution methods.

Importantly, existing data-driven approaches do not integrate prior information available on specific problems, for example, dependence in the decision variables or structure in the constraint matrix. This makes them computationally more intensive, and there is no guarantee that they are able to detect useful and obvious structures within the formulation. Our paper proposes data-driven approaches employing prior information in the decomposition of the problem and it demonstrates the applicability to large-scale optimization problems.

There have been past works in computing science that introduced approaches and algorithms for sub-problem decomposition within the framework of graph or network decomposition. The existing research focuses on the communication within the network, in such a way that communication between computing nodes was minimized (Hromkovič, 2013; Knobe, Lukas, & Steele, 1990; Nowak, 2003; Wolfe, Haghighi, & Klein, 2008). However, there are some key differences between those works and the proposed approaches in this paper. First, our goal for finding a decomposition is not to minimize communication but to minimize the number of constraints being dualized, in order to conserve as much structure of the original problem as possible, while decomposing the optimization problem. Within the framework of graph or network decomposition, each node in the network of this paper represents not a computing node but a decision variable or a group of decision variables.

Community detection has been applied to the Internet, citation networks, social networks, among others (Fortunato, 2010; Girvan & Newman, 2002). More details of community detection and its modularity measure can be found in Clauset, Newman, & Moore (2004), Newman & Girvan (2004), Newman (2004). There are different versions of community detection, and for their review, we refer the readers to Khaniyev et al. (2018). Community detection does not require the input of the number of clusters as input (unlike many unsupervised clustering algorithms such as k-means and k-medoids methods Xu & Tian, 2015; Xu & Wunsch, 2005), it does not have a parameter to tune (see Frey & Dueck, 2007 and Rodriguez & Laio, 2014), and it has been widely used in applications. We also note that the variable partitioning problem we consider has been formally formulated and shown to be NP-hard (Aykanat et al., 2004; Brandes et al., 2007; Bui & Jones, 1992; Iyadault, 2010), and the focus of this paper is to improve the empirical performance of parallel algorithms through variable partitioning.
3. Dual decomposition and sub-gradient method

Dual decomposition is a common technique for decomposing a large-scale optimization problem into smaller sub-problems (Bertsekas, 1995; Raffard et al., 2004; Terelius et al., 2011). Given a partition of decision variables, constraints that are over multiple groups of variables are relaxed and added to the objective function as penalty terms for violations, so that the Lagrangian relaxation is decomposable into smaller sub-problems. In this section, we briefly review the dual decomposition technique, followed by a parallel sub-gradient algorithm. We also analyze its convergence rate established in the existing literature and discuss why the convergence may be slow.

3.1. Transportation and resource allocation problems

Transportation problem is a general class of problems, in which commodities are transported from a set of sources to a set of destinations. Let $x_{ij}$ denote the matching variable from demand location $i \in I$ to supply location $j \in J$. Let $X$ denote the $|I| \times |J|$ matrix whose $(i, j)$ entry is $x_{ij}$ and $X_i$ denotes the ith row. The general transportation problem is given as follows.

\[
(\text{TP}) \min_{X \geq 0} \sum_{i \in I} \sum_{j \in J} w_{ij}x_{ij} \tag{1}
\]

s.t. $\sum_{j \in J} x_{ij} \geq m_i$ for $i \in I$, \hspace{1cm} (2)

$\sum_{i \in I} x_{ij} \leq s_j$ for $j \in J$. \hspace{1cm} (3)

where $m_i$ is the minimum demand that needs to be satisfied at each demand location $i \in I$, $s_j$ is the maximum capacity at each supply location $j \in J$, $w_{ij}$ is the cost associated with demand location $i$ getting one unit of goods from supply location $j$, $I_j$ is the set of supply locations that can serve demand location $i$, and $I_i$ is the set of demand locations that can be served by supply location $j$. In real applications where there is a large number of demand and supply locations, it is often assumed that each demand location can be served only by a subset of supply locations. For instance, in logistics, suppliers may have access only to a few demand locations due to regions of operations, or some demand locations are too far away. In this paper, we consider only continuous decision variables. For example, $x_{ij}$ may be a number of service hours assigned to demand location $i$ from supply location $j$.

3.2. Dual decomposition and parallel sub-gradient method

In this section we review a parallel sub-gradient method for (TP) with a straightforward partition of decision variables, a sub-problem for each demand location $i$. In order for (TP) to be decomposed for each $i$, all of the supply constraints (3) are relaxed and appended as penalties for their violation to the objective function. Let $\lambda_j \geq 0$ be the dual variable for each constraint in (3). The local sub-problem is written as follows:

\[
(\text{LR}_i) \min_{X_i \geq 0} L_i(X_i, \Lambda) = \sum_{j \in J} (w_{ij}x_{ij} + \lambda_j x_{ij}) \tag{6}
\]

s.t. $\sum_{j \in J} x_{ij} \geq m_i$.

A parallel sub-gradient algorithm for solving the Lagrangian dual is given as follows. 

Parallel Sub-gradient Algorithm

1. Choose a starting point $\Lambda^1$. Let $t := 1$ (first iteration).
2. Solve the local optimization problem $(\text{LR}_i)$ with $\Lambda = \Lambda^t$ for each demand location $i \in I$ to obtain $X_i^t$.
3. If a given stopping criterion is satisfied, stop. Otherwise, let $t := t + 1$, update the multipliers as below, and go to Step 2:

\[
\lambda^t_{j} = \max \{ \lambda^t_{j} + \alpha_t (x_{ij} - s_j), 0 \} \quad \text{for} \ j \in J. \tag{4}
\]

It is well-known that if the step-size $\alpha_t$ satisfies

\[
\sum_{t=1}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty, \tag{5}
\]

then the value of $g(\Lambda^t)$ converges to the optimal objective function value of (TP), where $g$ denotes the objective function of the Lagrangian dual (e.g., see Bertsekas, 1995). Moreover, the running average of the primal iterates $X^t$ becomes asymptotically optimal for (TP) as $t$ goes to infinity (Nedic & Ozdaglar, 2009a; Simonetto & Jamali-Rad, 2016).

3.3. Analyzing convergence rate of sub-gradient method

Convergence rates of sub-gradient methods have been established under various settings (Boyd, 2014; Goffin, 1977). We review a convergence rate result of sub-gradient methods with step-sizes satisfying (5) and discuss its slow convergence and our proposed approach to address it.

Since sub-gradient methods do not improve monotonically, it is common to keep track of the best solution up to the current iteration. Let $\Lambda^\text{best}$ denote the solution having the lowest $g$ value at the end of iteration $t$. Let $R$ be an upper bound on the distance between the initial dual solution and the set of optimal dual solutions, i.e., $\|\Lambda^1 - \Lambda^*\|_2 \leq R$ for any optimal solution $\Lambda^*$. Also, let $G$ be an upper bound on the norm of the sub-problems computed by the algorithm, i.e., $\|h^t\|_2 \leq G$, where $h^t = \sum_{i \in I} (x_{ij}^t - s_j)$ for $j \in J$. From Boyd (2014), we have the following upper bound on the optimality gap, which goes to zero as $t$ goes to infinity:

\[
g(\Lambda^\text{best}_t) - g(\Lambda^*) \leq \frac{R^2 + G^2 \|\alpha_t\|_2^2}{2 \sum_{t=1}^{\infty} \alpha_t} \tag{6}
\]

However, depending on the value of its numerator, the upper bound may converge to zero so slowly that it does not approach zero even at a large value of $t$ (e.g., hundreds of thousands). See Fig. 1 illustrating the significant difference in the convergence of the upper bound depending on the value of the numerator where $\alpha_t = \frac{1}{t}$. More importantly, the optimality gap itself may not approach zero even after a large number of iterations under the dual decomposition for each demand location. We emphasize that the slow convergence of the theoretical upper bound applies to any optimization problem, not limited to transportation problems used for illustration in this paper.

Next, we discuss possible ways to speed up the convergence of the upper bound. The upper bound contains the step size $\|\alpha_t\|_2$, an upper bound $R$ on the distance between the initial dual solution and the optimal dual solution set, and an upper bound $G$ on the magnitude of the sub-gradients. Adjusting the step size is an easy choice for accelerating sub-gradient methods, but it is specific to each application and purely empirical. Another important component that governs the behavior of the upper bound is the number of dualized constraints, because it equals the dimension of dual parameter vector $\Lambda$ and the dimension of the sub-gradients. Therefore, the number of dualized constraints is closely related to the magnitude of $R$ and $G$, and thus, directly affects the convergence of the upper bound.

In addition, each component of the sub-gradient at a primal solution $X$ is the violation of the corresponding dualized constraint...
at \( X \). Thus, the fewer dualized constraints are violated (and the smaller magnitude the violations have), the lower the magnitude of the sub-gradient is. This again emphasizes the importance of the number of dualized constraints. Dualizing more constraints leads to a more relaxed feasible region of the resulting Lagrangian relaxation. Then, the primal iterates obtained while running the sub-gradient method have more “room” to deviate from the original feasible region, thus allowing violations of more dualized constraints and also larger violations, which leads to higher magnitudes of the sub-gradients, and thus, a higher \( G \) value and slower convergence.

For the transportation problem and the dual decomposition introduced in the previous section, the intuition provided above is interpreted as follows. The Lagrangian relaxation of \( (TP) \) is obtained by dualizing all of the supply constraints \( (3) \), so the resulting relaxation differs significantly from the original problem. The sub-problem \( (LR_i) \) for demand location \( i \) is simply matching the demand of \( i \) to accessible supply locations where the values of the dual multipliers make the location \( i \) prefer some supply locations than others. Thus, the competition among demand locations for limited resources is only indirectly reflected via the dual multipliers. In other words, the level of decomposition is so fine that each sub-problem \( (LR_i) \) loses an important aspect of the original problem, which makes the overall convergence slow.

A partition of decision variables critically affects the computational performance of the sub-gradient method, and the goal of this paper is to develop a novel method to find a decomposition that speeds up parallel algorithms. In the context of dual decomposition, we aim at dualizing as fewer numbers of constraints as possible while taking the computational advantage of parallel computing. Consider decomposing \( (TP) \) into a given number of sub-problems by partitioning demand locations. The demand constraints \( (2) \) are decomposable by demand locations, but the supply constraints \( (3) \) are not. Given a partition of demand locations, those supply constraints involving demand locations in multiple groups need to be dualized in order for the remaining constraints to be decomposable. Herein we define two demand locations to be connected if and only if there exists a supply location that can serve both demand locations, i.e., they appear together in the capacity constraint of the supplier. Then, finding a decomposition with a minimal number of dualized constraints translates into finding a partition in which demand locations in the same group are closely connected, and those from different groups are loosely connected. We expand on this idea in the next section after discussing two other partitioning methods which are based on external knowledge.

4. Partitioning methods and block dual decomposition

In this section, we introduce a novel framework for parallel optimization, consisting of two steps:

- **Step 1**: Variable partitioning; and
- **Step 2**: Block dual decomposition.

The proposed approach uses the structure of constraints to speed up the convergence of parallel sub-gradient methods. We first illustrate the approach using the transportation problem, but we also introduce a general version of the framework in Section 4.3.

4.1. Step 1: variable partitioning

In this section, we discuss three approaches to partitioning decision variables. The three approaches apply generally to networks with an established structure, but we will specifically introduce the approaches in the context of geographically or spatially structured networks, such as in transportation problems. The output of these approaches consists of blocks of variables; those variables assigned to different blocks are assumed to be de-coupled while those within the same block are assumed to be coupled within the parallel optimization framework.

4.1.1. Prior knowledge

In many applications, particularly in transportation problems, prior knowledge could suggest how the decision variables can be partitioned into different blocks. In cases where variables correspond to geographic locations, a grouping of the variables can be based on geographical sub-areas, such as county, census tracts, health districts, etc.

4.1.2. Clustering

Clustering algorithms such as \( k \)-means can be used to obtain a partition of decision variables, given that a notion of similarity between decision variables (or between groups of decision variables) is defined. In applications to transportation problems, the Euclidean or travel distance can be used as a similarity measure for clustering demand locations. For a given number of clusters \( K \), a clustering algorithm iteratively finds a partition of demand...
locations into $K$ clusters in such a way that each demand location belongs to the cluster whose center is the closest in distance. Another view of this approach is partitioning the network space into Voronoi cells, where, loosely speaking, the demand locations in each of the cells are close in distance or other similarity measures. In Section 5.1.2, we discuss how the granularity of the partition, i.e., the number of clusters, affects the performance of the sub-gradient algorithm.

### 4.1.3. Community detection

In this section, we consider an approach that uses the structure of the optimization formulation itself to partition decision variables into blocks. We first build a network graph representing how decision variables (or groups of decision variables) are related through constraints. Then, we apply a community detection algorithm to find a partition informed by the structure of constraints.

In the context of the transportation problem, we first build a network graph of demand locations. Consider a graph of $n$ nodes, with each node representing one demand location. Two nodes are connected by an edge if the corresponding demand locations are connected, that is, the two demand locations have access to a common supplier. The edge is weighted by the number of suppliers that can serve both of the locations, i.e., the number of constraints the two demand locations appear together. To this network, we apply a community detection algorithm to identify communities of demand locations where those in the same community are densely connected, and those in different ones are sparsely connected. Then, we decompose the optimization problem according to the communities.

Among various algorithms developed in the community detection literature, we use the fast hierarchical agglomeration algorithm proposed by Clauset et al. (2004). The computational complexity of the algorithm is linear in the size of the network for many real-world networks. We briefly explain how the algorithm works below.

The community detection algorithm is based on a measure of a partition called the modularity, which evaluates how dense connections are within communities and how few there are between communities (Newman & Girvan, 2004). Before defining the measure, we introduce some notation. An $n$-by-$n$ weighted adjacency matrix $C$ is defined as

$$C_{uv} = \begin{cases} e_{uv} & \text{if nodes } u \text{ and } v \text{ are connected,} \\ 0 & \text{otherwise,} \end{cases}$$

where $e_{uv}$ is the weight of the edge $(u, v)$. Consider a partition of the nodes and for a node $u$, let $c_u$ denote the community to which $u$ belongs. Let $k_u = \sum_v C_{uv}$ be the sum of weights of all edges in the graph and let $k_u = \sum_v C_{uv}$ be the sum of weights of all edges from $u$. Then, the modularity of a partition is defined as:

$$Q = \frac{1}{2k} \sum_{u,v} \left( C_{uv} - \frac{k_u k_v}{2k} \right) \delta(c_u, c_v).$$

(7)

In the above definition, the fraction $k_u k_v / (2k)$ is the expected number of edges between $u$ and $v$ where $k$ edges are randomly assigned between nodes. Thus, the modularity measures how strong the community structure is over a random assignment of edges. In practice, networks with the modularity greater than 0.3 appear to indicate significant community structure (Newman, 2004).

The community detection algorithm starts with a trivial division where each of the demand locations forms a community. Then, it repeatedly joins two communities that results in the biggest increase of the modularity, until it reaches a partition where none of the joint operations improves the modularity score. More details of the algorithm can be found in Clauset et al. (2004).

### Remark 4.1

We briefly compare an overview of the computational complexity of our proposed community detection approach with that of Khaniyev et al. (2018), both applied to the transportation problem. The input for our approach is a graph with $l$ nodes and $O(l^2)$ edges while that of Khaniyev et al. (2018) is a graph with $l + l + j$ nodes and $O(l^2 + l^3)$ edges. Both our approach and Khaniyev et al. (2018)'s rely on running the community detection algorithm on these graphs. Since the size of the graph in the approach of Khaniyev et al. (2018) is larger, this makes it computationally more demanding, and intractable for large scale transportation instances which is the focus of our paper.

### 4.2. Step 2: block dual decomposition

Each block in the output of the aforementioned variable partitioning algorithms may include multiple demand locations. Thus, the corresponding dual decomposition yields sub-problems that include blocks of demand locations, and thus we call the proposed approach block dual decomposition. However, we emphasize that a grouping of decision variables varies by a partitioning method and that the community detection method groups those that are closely connected in the network graph representing the structure of the constraints. This characteristic of our approach makes the resulting sub-problems keep as much structure of the original optimization problem as possible, which is critical for the performance of the parallel sub-gradient method, as explained in Section 3.3 and empirically shown in Section 5.

Let $b_p$ for $p = 1, \ldots, P$ be the partition of demand locations given by the partitioning algorithms; thus satisfying $b_p \cup b_q = \emptyset$ for $b_p \neq b_q$. Let $b_p$ be the set of suppliers that can serve the demand locations in $b_p$ (e.g., within a pre-specified distance). Note that the set $b_p$’s may not be disjoint as opposed to $b_q$’s. For each block $b_p$, the suppliers that can serve only the demand locations in the block are said to be interior suppliers of block $b$, denoted as $f_{in}^p$, and the suppliers that are not interior suppliers but can serve a demand location in $b_p$ are called boundary suppliers of block $b_p$, denoted as $f_{out}^p$. Let $f_{in} = \bigcup_p f_{in}^p$ and $f_{out} = \bigcup_p f_{out}^p$. Note that $f_{in}$ for $b = 1, \ldots, B$ are disjoint. For a demand location $i$, let $b(i)$ denote the block to which $i$ belongs.

Consider the following Lagrangian relaxation of (TP):

$$\text{(BLR)} \quad \min_k L(X, \Lambda) = \sum_{i, l} w_{ij} x_{ij} + \sum_{j \in b} \lambda_j \sum_{i \in l} x_{ij} - s_j$$

s.t. $$\sum_{j \in b} x_{ij} \geq m_i \quad \text{for } i \in I,$$

$$\sum_{i \in l} x_{ij} \leq s_j \quad \text{for } j \in f_{in},$$

$$X \geq 0.$$

(11)

Note that among the supply side constraints (3), only those corresponding to boundary suppliers were dualized in (BLR). Consequently, a fewer number of dual variables are needed than in the previous section. By following similar steps to those of the previous section, (BLR) is decomposed as follows:

$$\text{(BLRb)} \quad \min_k \sum_{i, l} w_{ij} x_{ij} + \sum_{j \in b} \sum_{i \in l} \lambda_j x_{ij}$$

s.t. $$\sum_{j \in b} x_{ij} \geq m_i \quad \text{for } i \in b_b,$$

$$\sum_{i \in l} x_{ij} \leq s_j \quad \text{for } j \in f_{in},$$

$$X_i \geq 0 \quad \text{for } i \in b_b.$$
The resulting parallel subgradient algorithm is as follows. **Parallel subgradient algorithm with block dual decomposition**

1. Choose a starting point: $\Lambda^1 = 0$. Let $t := 1$.
2. Solve the local optimization problem (BLR$_b$) for each demand block $b$ to obtain $X_b^t$ for $i \in h_b$.
3. If (some stopping criterion) is satisfied, stop. Otherwise, $t := t + 1$, update the multipliers as below, and go to Step 2:

   $$
   \lambda^{t+1}_j = \max\{\lambda_j^t + \alpha_t \left( \sum_{i \in A_j} x_{ij} - s_j \right), 0\} \text{ for } j \in j^\text{int}.
   $$

4.3 A general approach

We have illustrated details of the block dual decomposition with community detection under the transportation problem setting. In this section, we present a similar approach, but with a broader applicability. First we present our approach for the case for which no prior information is available, and then generalize it to a setting with prior information. Consider the following optimization problem:

$$
\begin{align*}
\min f(x) \\
\text{s.t. } Ax = d,
\end{align*}
$$

where $A \in \mathbb{R}^{m \times n}$ and $f : \mathbb{R}^n \to \mathbb{R}$ is decomposable for each component of $x$, i.e., $f(x) = \sum_{i=1}^n f_i(x_i)$. For this general formulation, we illustrate how our approach can be applied to find a partition of decision variables for which the corresponding dual decomposition dualizes a minimal number of constraints.

Construct a graph in which each node represents a decision variable. Two nodes are connected if the two decision variables appear together in a constraint, and the edge is weighted by the number of constraints they appear together. Note that in this section, each node represents a decision variable as opposed to the previous section, where each node corresponds to a demand location for the transportation problem. Then, we apply the community detection algorithm to the graph in order to identify a partition of decision variables where connections within each group are dense, but those between groups are sparse. A weighted adjacency matrix $C$ is constructed as follows. We first form an indicator matrix $\tilde{A} \in \mathbb{R}^{m \times n}$ such that for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$,

$$
\tilde{A}_{ij} = \begin{cases} 
1 & \text{if } A_{ij} \neq 0 \\
0 & \text{if } A_{ij} = 0.
\end{cases}
$$

Thus, $\tilde{A}_{ij} = 1$ if $x_i$ appears in constraint $j$. Then, an $n$-by-$n$ weighted adjacency matrix $C$ is defined as $C = \tilde{A} \tilde{A}^T$, thus $C_{uv}$ is the number of times variables $x_u$ and $x_v$ appear together in constraints, for $u = 1, \ldots, n$ and $v = 1, \ldots, n$. Then, the modularity score of a partition is computed by using this $C$ matrix as (7), and the community detection algorithm is applied. If the objective function is decomposable by groups of decision variables instead of each variable, then the algorithm mentioned above can be trivially extended by treating each group of variables as one node in the graph.

We now discuss how prior information may be incorporated in this general framework. For this purpose, we first discuss how the prior information may be presented formally. For the transportation problem, prior knowledge appears as a geographical partitioning of the variables. By extending this notion to the general framework, the prior information should appear as an initial partitioning of variables, for example, communities in social networks or regional grids in an electric network. Now, consider $x_i$ as a block of variables in this initial partition. Note that $A_{ij}$ represents a vector, consisting of all elements of row $i$ and the block of variables $j$ within the constraint matrix. The definition of the indicator matrix $\tilde{A} \in \mathbb{R}^{m \times n}$ remains the same with the only difference that $A_{ij}$ is compared with the zero vector of appropriate dimension. The rest of method regarding construction of the weighted adjacency matrix $C$ and the community detection algorithm remains unchanged. Finally, note that if no prior information is available, we may use the most trivial initial partition in which each variable is considered as a single block. For this case, our approach using this trivial initial partition coincides with the approach presented in the earlier part of this section.

5. Numerical results

In this section, we present experimental results using transportation problems based on a real application. To illustrate generality, the approach provided in Section 4.3 was applied to a variety of multi-dimensional knapsack problems (MKPs) in Appendix A.

Our methodology was implemented in Julia, a high-performance programming language for numerical computing (Bezanson, Karpinski, Shah, & Edelman, 2012), along with Gurobi for optimization. Unless mentioned otherwise, we used Intel Core Haswell Processors with 16 GB RAM on a Linux server with X86-64 bit architecture.

5.1. Case study: transportation problem in healthcare

5.1.1. Problem setup

In the experimental study, we generated problem instances based on an optimization model from a real application: matching children in need of primary care to healthcare providers in Georgia. The optimization problem takes the form of a general transportation problem (TP), and it minimizes the total distance that patients have to travel to receive care.

In this problem setting, each demand location is a census tract, and each supply location is a healthcare provider. Census tracts are used as proxies of communities, and they form a contiguous division of a state. The patient population is aggregated at the census tract level using the geographic division established in the 2010 SF2 100% census data. To compute the number of children in each census tract, the 2012 American Community Survey data were used. Providers’ practice location addresses, i.e., supply locations, were obtained from the 2013 National Plan and Provider Enumeration System (NPPES). More details about the application problem can be found in Gentili, Serban, Harati, O’Connor, & Swann (2017).

In the optimization problem, the decision variable $x_{ij}$ denotes the number of children in demand location $i \in I$ assigned to supply location $j \in J$; $w_{ij}$ is the travel distance between the demand location $i$ and supply location $j$; $m_i$ is the minimum number of patients needed to be served at demand location $i$; $s_j$ is the maximum number of patients supply location $j$ can accommodate. We allow the variables $x_{ij}$ to be fractional for the computational tractability of the problem, and also because the number of children to be assigned from each location is typically large (approximately 2500–8000 children). There are 1955 demand locations and 3157 supply locations.

Using this optimization problem, we created problem instances with different sizes to evaluate the computational complexity of the proposed methodology with the problem size using the following method. First, we divided the map of Georgia into 50 blocks, 10 horizontally by 5 vertically, based on the longitudinal and latitudinal coordinates. Then, we counted the number of census tracts and provider locations in each block. For each block, we constructed the empirical distributions of the demand ($m_i$’s) across the census tracts and of the supply capacities ($s_j$’s) of providers in the block.

We generated problem instances for a given number of demand and supply locations assumed for the optimization problem as fol-
lows. We determined the number of demand locations in each block in a way that the numbers of demand locations in different blocks of a generated instance are proportional to the numbers of demand locations in blocks of the original problem. In the same way we determined the number of supply locations in each block. Positions of demand and supply locations in each block were sampled randomly from the uniform distribution over the block. For each demand or supply location, the amount of demand or capacity was sampled from the empirical distributions of the block for demand or supply, respectively. In addition, a demand location \( i \) was said to have access to a supply location \( j \) if the distance \( w_{ij} \) between them was less than or equal to a given threshold \( d_{\text{max}} \). By changing the threshold \( d_{\text{max}} \) on the traveling distance, we were able to adjust the connectivity between demand and supply locations, thus changing the connectivity of the network. A lower \( d_{\text{max}} \) indicates a sparser network.

5.1.2. Partitioning methods

In this section, we applied the four different techniques described in Section 4.1 to derive variable partitions. We compared the four methods with a baseline partitioning method. The five approaches are described below.

Geo-HD: The first technique partitions the problem completely based on prior knowledge: we grouped the census tracts based on their corresponding public health district affiliation. The Georgia Department of Public Health collaborates with the public health districts throughout the state, while each health district oversees the operation of its affiliated health departments. There are 10 health districts in Georgia.

Geo-KM: The second technique is a clustering-based method (k-means) that operates on geographical information for partitioning. This is the only method for which a user should choose the number of blocks.

Opt-KKE: The third technique (called KKE algorithm herein) is an iterative approach, which was introduced by Khaniyev et al. (2018). This approach is applied to the constraint matrix of the optimization problem. Its primary component is the use of a community detection methodology in lieu of graph/hypergraph associated with the constraint matrix, where each row of the matrix is represented by a row, and non-zero elements of the matrix correspond with the arcs. Then, they use the community detection approach to partition nodes of the graph in an iterative manner. Compared to the previous papers in the literature, Opt-KKE alleviates the drawback of needing the number of clusters.

Opt-CD: The fourth approach is our community detection approach. Variable partitioning given by the community detection based on the constraint structure of the optimization problem. To note, the KKE approach is different from the one introduced in this paper in that it does not integrate any additional information about the geographic dependence.

Baseline: We use a fifth baseline partition scheme, in which each demand locations becomes one of the communities. This is the common approach considered in dual decomposition problems.

5.1.3. Optimization setup

We first implemented the sub-gradient method with the variable partitioning methods in a sequential computing fashion, that is, all sub-problems in each iteration are solved sequentially using one computing node. In addition, we implemented a parallel version of the sub-gradient method, which solves the sub-problems in parallel at each iteration using certain number of computing cores. The step size \( \alpha_t \) was chosen to be \( c/t \), where \( c \) is a constant scaling factor. When the problem size is not extremely large, we are able to compute the optimal value (i.e. without any partitioning) with a maximum of 100 GB RAM (this is the largest computing resource we can get). Given the optimal value, we can calculate the optimality gap (i.e. percent difference between the optimal value and the solution to the optimization problem from each iteration) and thus terminate the sub-gradient algorithm given a optimality gap (e.g. 5%). We achieved load balancing through feeding the sub-problems into the distributed framework based on their sizes such that sub-problems with similar sizes are distributed around the same time.

For all partition methods, we considered different values of the scaling factor \( c = 1/10, 1/50, 1/25, 1/80, \text{ and } 1/100 \), but all of the methods had the fastest convergence for the same \( c \) value at 1/25, which we used for the comparison. The performance measures for the comparison included the number of iterations and the time in seconds (the CPU time for the sequential version and the wall clock time for the parallel version) required to reach a certain optimality gap percentage.

5.1.4. Experiments related to partitioning

We designed different experiments to compare and contrast the optimization and computing performance of five different partitioning methods through analyzing the results of the sub-gradient method using the partitioning methods.

Comparison between community-detection-based partitioning methods Two of the partitioning methods we introduced in Section 5.1.2 are based on the community-detection algorithm: Opt-KKE and Opt-CD. Table 1 shows some detailed comparisons between these two community-detection-based approaches. The runtime is collected from the parallel implementation of sub-gradient method, which terminates when the optimality gap falls below 5%. Since the Opt-KKE is an iterative approach, it naturally takes more time compared to the single-iteration Opt-CD approach. See the comparison results in Table 1 for all instance sizes. As provided by this results, the Opt-KKE is not a practical partitioning method when the instance size becomes large because the runtime increase exponentially with the instance size. In addition, we observed that the community detection algorithms may result in a partition with many small communities or blocks. Such extreme results are more frequently when using Opt-KKE; the majority of the communities have small size (i.e. number of demand locations) with only one or few very large size partitions. In problem instances 2 and 3, all except one block are small-sized blocks. The largest block resembles the original problem instance, making Opt-KKE not applicable for large problems due to the long computing time when solving the problem without variable partitioning. Therefore, we will only consider Opt-CD as a representative community detection partitioning method for the remainder of the experimental results in this paper.

Comparison among different partitioning methods We compare the performance of the four remaining partitioning methods, Baseline, Geo-HD, Geo-KM, Opt-CD, using the sub-gradient method terminated at 5% optimality gap.

We first generated a problem instance with 1000 demand locations, 1000 supply locations, and \( d_{\text{max}} = 20 \) (miles) generated as explained in Section 5.1.1. The instance has 67,163 decision variables in total. We set \( K = 6 \) for the clustering-based approach to be consistent with the number of blocks yielded from the community detection algorithm. Fig. 2 shows the comparison for this problem instance with the four partitioning methods and how their optimality gap progressed as a function of the number of iterations for sequential implementations of sub-gradient method. For any of the three non-baseline partitioning methods, their sub-gradient method required significantly fewer iterations to achieve the same optimality gap than the baseline partitioning method. The Baseline method reached the stopping criterion after 17,231 iterations. On the other hand, Geo-HD, Geo-KM, and Opt-CD finished after 3234, 1965, and 1881 iterations, respectively. Under the parallel implementation, the Baseline took about 9.6 hours, but the Geo-HD, Geo-
Table 1
Detail Comparison between Opt-KKE and Opt-CD.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Instance 1</th>
<th>Instance 2</th>
<th>Instance 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Size</td>
<td># Demand Locations</td>
<td># Supply Locations</td>
<td># Variables</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>26,444</td>
<td>67,163</td>
<td>228,000</td>
</tr>
<tr>
<td>Runtime (sec)</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>0.7106</td>
<td>20.2148</td>
<td>20.2148</td>
</tr>
<tr>
<td></td>
<td>3.2521</td>
<td>11477.87</td>
<td>11477.87</td>
</tr>
<tr>
<td># Blocks</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td># Blocks with size ≥ 40</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td># Blocks with size ≤ 3</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

Fig. 2. Comparison among various variable partitioning methods for 1000 demand locations and 1000 supply locations, in log scale.

Table 2
Size of different problem instances.

<table>
<thead>
<tr>
<th>Table 2</th>
<th>Instance 4</th>
<th>Instance 5</th>
<th>Instance 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Size</td>
<td># Demand Locations</td>
<td># Supply Locations</td>
<td># Variables</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td>26,444</td>
<td>67,163</td>
<td>228,000</td>
</tr>
<tr>
<td># Blocks with Geo-HD</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td># Blocks with Geo-KM</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td># Blocks with Opt-CD</td>
<td>Opt-CD</td>
<td>Opt-KKE</td>
<td>Opt-KKE</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 3
Comparison on reaching 5% optimality gap for problem instances with varying sizes.

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Instance 1</th>
<th>Instance 2</th>
<th>Instance 3</th>
</tr>
</thead>
<tbody>
<tr>
<td># Iterations</td>
<td>Baseline</td>
<td>Block with Geo-HD</td>
<td>Block with Geo-KM</td>
</tr>
<tr>
<td></td>
<td>4,375</td>
<td>568</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>17,231</td>
<td>3233</td>
<td>1965</td>
</tr>
<tr>
<td></td>
<td>&gt;80,000</td>
<td>3164</td>
<td>4122</td>
</tr>
<tr>
<td>Run Time (in Seconds)</td>
<td>Sequential Baseline</td>
<td>Sequential Block Geo-HD</td>
<td>Sequential Block Geo-KM</td>
</tr>
<tr>
<td></td>
<td>1,723</td>
<td>212</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>24,532</td>
<td>7546</td>
<td>3846</td>
</tr>
<tr>
<td></td>
<td>&gt;300,000</td>
<td>12,959</td>
<td>17,988</td>
</tr>
<tr>
<td></td>
<td>Sequential Block Opt-CD</td>
<td>Parallel Block Geo-HD</td>
<td>Parallel Block Geo-KM</td>
</tr>
<tr>
<td></td>
<td>133</td>
<td>83</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>5115</td>
<td>3210</td>
<td>1577</td>
</tr>
<tr>
<td></td>
<td>6016</td>
<td>6712</td>
<td>9144</td>
</tr>
<tr>
<td></td>
<td>Parallel Block Opt-CD</td>
<td>52</td>
<td>2243</td>
</tr>
<tr>
<td></td>
<td>2956</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

KM, and Opt-CD took 53, 26, and 37 minutes, respectively. Therefore, we can conclude that solving the problem instance with any of the variable partitioning methods would perform better than that without partition.

Table 2 shows the size of additional problem instances, along with the number of blocks from the three partitioning methods. Table 3 shows comparisons on the rate of convergence in the number of iterations and the computing time using various partitioning methods. In Table 3, as the size of the problem increases, each algorithm takes more iterations and more time to reach 5% optimality gap, and the gap between the methods increases. The sequential implementation of the three sub-gradient methods achieved faster convergence in both the number of iterations and the run time than the Baseline by a large margin. The parallel implemen-
tations of the three block methods using three computing nodes yielded 2-2.5 times speedup compared to their sequential counterparts.

Comparison under problem instances with different connectivity
The sparsity of the constraint matrix of the optimization problem also affects the performances of each of the four partitioning methods. This sparsity is related to the connectivity of the network graph (i.e., how connected among supply and demand locations). We constructed problem instances with 1500 demand locations and 500 supply locations, but different values of \( d_{\text{max}} = 20, 25, \) and 30 in miles. Recall that \( d_{\text{max}} \) is the maximum distance to travel and that a supply location is accessible from a demand location if the distance between them does not exceed \( d_{\text{max}} \). Thus, the larger \( d_{\text{max}} \) is, each supply constraint involves more decision variables. In the network of demand locations for community detection (described in Section 4.1.3), two demand locations are connected if they share an accessible supply location, and thus, a larger value of \( d_{\text{max}} \) implies a more dense network. Fig. 3 compares the baseline and the three approaches using block partitioning up to 2500 iterations for the three instances with different connectivity. For the three values of \( d_{\text{max}} = 20, 25, \) and 30, each demand location had access to 51, 68, and 84 providers on average, respectively.

We observe that the approaches using any non-baseline partitioning methods perform better than the baseline consistently for different values of \( d_{\text{max}} \), and the gap of performance gets larger as the \( d_{\text{max}} \) increases. For \( d_{\text{max}} = 30 \) at the 2500th iteration, the optimality gap is 39% for the baseline, and 32%, 19%, and 15% for the block with Geo-HD, Geo-KM, and Opt-CD, respectively. For \( d_{\text{max}} = 25 \), the optimality gap is 31% for the baseline, and 27%, 9%, and 10% for the block with the three partition methods, respectively. For \( d_{\text{max}} = 20 \), the optimality gap is 19% for the baseline, and 11%, 2%, and 6% for the block with the three partition methods, respectively. We also observe that the optimality gaps of all approaches increase as the network becomes denser. Note that although \( k \)-means seems to produce better results among the three in most settings, it highly depends on the inherent problem structure. Thus the performance of different partition methods should be compared on a case-by-case basis.

The effect of the network structure on the performance can be explained geometrically as follows. For a larger value of the threshold, each provider is accessible from more demand locations, and thus, each provider constraint contains more decision variables. In that case, dualizing each provider constraint results in a bigger change on the feasible region in the following sense. For example, consider the following two relaxations: relaxing \( x_1 + x_2 \leq 1 \) from \( \{x_1, x_2 \} | x_1 + x_2 \leq 1, x_1 \geq 0, x_2 \geq 0 \} \) and relaxing \( x_1 \leq 1 \) from \( \{x_1, x_2 \} | x_1 \leq 1, x_1 \geq 0, x_2 \geq 0 \}. \) The former can be viewed to yield a bigger change than the latter. In this sense, when \( d_{\text{max}} \) is larger, dualizing each provider constraint causes a bigger change on the feasible region. Moreover, note that the baseline partitioning dualizes more provider constraints than the proposed approach. Therefore, as \( d_{\text{max}} \) increases, the discrepancy between the feasible regions of the Lagrangian relaxation and of the original problem becomes more significant for the baseline than it does for the block approaches. Thus, when \( d_{\text{max}} \) increases (i.e., the network gets denser), the baseline performs more poorly as compared to the block approaches.

Comparison under very large problem instance
To further demonstrate the application of our approach in real-life scenarios, we examined its performance using extremely large instances (large number of demand locations and supply locations). We created large instances based on the same approach as described in Section 5.1.1. Instead of the state of Georgia, we chose California because it has the largest population among all states in the U.S. Using California data and the same instance generation techniques, we created an instance containing 32,000 demand locations and 28,500 supply locations, resulting in 16,701,324 pairs of demand and supply locations within the threshold \( d_{\text{max}} = 20 \) miles.

We attempted to solve this large instance using Gurobi without any decomposition using a personal laptop with Intel Core i5-6300U CPU and 8 GB RAM using the sequential framework. This computation failed due to lack of memory allocation (i.e., RAM). Thus, this instance demands higher computing power, reduced computational load by using decomposition, or both.

Decomposing the problem via either health district information or using community detection algorithm enable the parallel implementation of the sub-gradient method to solve the optimization model successfully under the same computing resource. However, the computational cost of running a community detection algorithm on such a large instance is nontrivial; we observed that it
could use over 100 GB RAM and take over a day to find communities, which is definitely not practical under limited computing resource. Thus, we focus on decomposing the original model using health-district information. Also, by monitoring the run time of different partitions, we identified the bottleneck to be the largest-sized sub-problem. For example, in the decomposition obtained by health districts, the bottleneck is the county with the largest number of demand locations. If the computing resource of a single node cannot solve the largest sub-problem, the whole solution algorithm cannot be run.

Conclusions for experiments related to partitioning While we are not able to determine a single best partitioning approach that out-performs the others under all scenarios, we are able to gain some insights from the different experiments above. From “Comparison between community-detection-based partitioning methods”, we conclude that even though both Opt-KKE and Opt-CD are based on community-detection algorithm, while Opt-KKE is one of the best decomposition algorithms, Opt-KKE does not perform well in our application because it takes significantly longer to execute, hence not practical for large problem instances as illustrated in Remark 4.1. From “Comparison among different partitioning methods”, we see that solving the problem instance with some partitioning methods is better than solving it with the no partitioning (i.e. Baseline).

From “Comparison under problem instances with different connectivity” to “Comparison among different partitioning methods”, we note that the gap in performance gets larger as the problem instance becomes more complex (i.e. constraint matrix more dense). Each of the three partitioning methods had their own advantages and disadvantages. Geo-HD requires no further computation, but it needs the prior knowledge about health-district. Geo-KM requires the location information, the number of clusters, and the computation of partitions. The determination of the best number of clusters for Geo-KM requires special attention, which we will discuss in “Granularity of partitioning” below. Lastly, Opt-CD requires only the computation of partitions as it operates on the constraint matrix of the optimization problem. When facing a very large problem instance, the computing resource might become the decision factor. In “Comparison under very large problem instance”, we prefer using Geo-HD over the other two for this exact reason.

5.1.5. Experiments related to sub-gradient method

In this section, we design experiments on various factors that affect the performance of the sub-gradient method.

Granularity of partitioning The granularity of a partition is critical for the performance of the sub-gradient method because it affects the number of iterations for the sub-gradient method to converge and the level of difficulty of the sub-problems. We want to analyze this trade-off between granularity of a partition and the performance of sub-gradient method in a controlled manner. In an extreme case where there is only one block, the sub-gradient method takes simply one iteration, but the sub-problem (which is the original problem) is the most complex compared to that of any other partition. As a decomposition becomes finer, the resulting sub-problems become smaller, but the sub-gradient method requires more iterations to converge as we explained in Section 3.3. Among the five partitioning methods considered in this paper (i.e. Geo-HD, Geo-KM, Opt-KKE, Opt-CD, Baseline), the clustering method (i.e. Geo-KKE) is the only one for which a user chooses the number of blocks.

Fig. 4 illustrates this trade-off for the application considered. We generated a problem instance with 500 demand locations and 500 supply locations, and the clusters were obtained by using the k-means algorithm. The figure shows the CPU time to solve the largest sub-problem on average over different iterations and the number of iterations for the sub-gradient method to converge, for varying numbers of clusters. Note that in a synchronous parallel computing framework, the largest sub-problem is likely to be the bottleneck in each iteration. The time to solve the largest sub-problem over different iterations was similar. When k is greater than 9, the performance of the sub-gradient method becomes almost equivalent to baseline partitioning, which converges in 2437 iterations. This figure shows that the performance of the clustering-based partitioning method is affected heavily by the number of clusters and indicates that an optimal number of clusters must be determined for a given problem instance.

Computing cores We also investigated how the computational performance changes as the number of computing nodes increases. This analysis was motivated by the fact that the baseline decomposition results in a much larger number of sub-problems than the compared approaches. The baseline can distribute the sub-problems potentially into a large number of computing nodes, which may result in a computational advantage. We randomly generated eight instances with 1000 demand locations and 1000 supply locations using Georgia data with $d_{max} = 30$ miles. For each instance, we used the baseline partitioning and ran the sub-gradient method up to 15% optimality gap, using 1, 2, 4, 8, or 16 cores. When we varied the number of computing nodes for the same instance, the optimality gap after each iteration remained the same; this is because the only difference was how the sub-problems were distributed over computing nodes. Subsequently, we compared only the average run time per iteration. This set of experiments were executed on a Linux x86-64 architecture with a total of 12 physical cores and 24G RAM using Xeon E5645 CPUs, which can offer up to 24 parallel processes through the hyperthreading.

Fig. 5 shows the average run time per iteration with different numbers of computing cores for eight random instances. The run times of the instances at each number of cores are similar to each other, and a convex decreasing trend is clearly shown. We fitted a model $r = \exp(1/(0.5317 + 0.1194c)^2)$, where r is the average run time and c is the number of cores. The model fit has an $R^2 > 0.95$. The model and the figure demonstrate that the computational gain obtained by increasing the number of cores decays quickly as the number of cores increases, in other words, most of the computational gain of parallel computing is achieved at with a small number of cores.

Largest sub-problem size The performance of parallel sub-gradient method is affected by its bottleneck: the size of the largest sub-problem. Using only the health-district information to decompose the original model generated a decomposition in which
We value the largest sub-problem has 9248 demand locations. We obtained finer decompositions by enforcing an upper limit on the size of the largest sub-problem as follows. For each sub-problem, if the size exceeds the upper limit, we use the k-means algorithm to partition the sub-problem into two smaller ones. We used different values for the maximum sub-problem size, 100, 200, 500, and 1000. For this experiment, we used 16 parallel processes under the same Linux x86-64 architecture as the previous experiment of Fig. 5. It is also possible to solve the instance with these decompositions using a personal laptop with Intel Core i5-6300U CPU with 8 GB RAM we previously used, but it takes long time. For example, with the maximum sub-problem size 100, each iteration took more than an hour. To compute optimality gaps, we obtained the exact optimal value of the instance by solving it using Gurobi in a server with 100 GB RAM, which is the largest computing resource we can get. We should note that 100 GB RAM is very difficult to get in any non-high-performance computing setting.

Table 4 shows the number of iterations to reach 30% optimality gap, the average run time per iteration when the computation is distributed over 16 cores, and the average run time when a single core is used, for different limits on the size of the largest sub-problem. We observe that when the maximum size of sub-problems is smaller (that is, a finer decomposition), it requires more iterations. Also, the average run time per iteration increases as the maximum size of sub-problems goes up, with one exception of the parallel case with the maximum size 200. The results show that our approach with higher computation power could solve the large instance; the same problem could not be solved by a modern solver without decomposition when not employing a high-performance computing setting with large RAM. For each decomposition, we also compared the run time between the sequential implementation and the parallel one. The progress of the optimality gap at each iteration is the same between the two approaches because both use the same decomposition, so we simply compared the average run time per iteration. We observed that the run time of each iteration stabilizes after the first few iterations and does not vary much afterwards; this is because the sizes of the sub-problems solved in different iterations are similar. Thus, for the sequential approach, we ran it for 30 iterations and computed the average run time per iteration. The results in Table 4 show that parallel computing gives a significant gain in the run time, but the gain is much less than 1/16 when using 16 cores.

Parallel vs. sequential As we can see from “Largest sub-problem size”, sequential implementation takes significantly more time compared to the parallel implementation when other settings are the same (e.g., termination, computer architecture, etc.). However, unarguably, parallel implementation demands more computing resources, making it not practical when facing a large problem instance, such as the one described in “Comparison under very large problem instance”.

Conclusion on experiments related to sub-gradient method While we cannot conclude the single best procedure to execute the sub-gradient method, we identified several factors to be considered before starting the optimization: the results from partitioning “Granularity of partitioning” and “Largest sub-problem size”) and the available computing resources “Computing cores” and “Parallel vs. sequential”.

6. Conclusion

In this paper, we proposed a framework for determining a partition of decision variables towards decomposing a large-scale optimization problem, in a way that improves the performance of parallel optimization methods.

We first showed that a partition of decision variables in dual decomposition could be crucial for the empirical performance of parallel sub-gradient methods. Then, we proposed methods for finding a partition of variables that minimizes the number of constraints being dualized. We demonstrated that integrating knowledge about the structural features about the optimization problem improves detecting meaningful partitions, making the problem amenable to distributing computing, hence reducing the computational effort in large-scale optimization problems.

Specifically, within a real-world transportation problem, we demonstrated that the problem instance itself as well as the available computing resources are decisive in selecting the most suitable procedures for partitioning the decision variables and executing the sub-gradient method. Each of the proposed partitioning methods compared in this study have their own strengths and weaknesses, which makes them more suitable under certain scenarios.

For example, when we have limited computing resources, partitioning the decision variables based on prior knowledge alone would be preferable as it requires no additional computing. The sequential computing framework for optimization will become more practical as it requires less computing resources compared to parallel computing. However, the parallel implementation outperformed the sequential implementation in terms of running time.

When comparing different data-driven methods for partitioning variables, the community-detection or the clustering method, we found that the community detection method is computationally infeasible for high-dimensional optimization problems (e.g., California example). Between the two community-detection-based algorithms compared in this study, the one introduced by Khaniyev et al. (2018) is computationally expensive compared to community-detection approach introduced in this paper. As for the spatial clustering approach, different number of clusters will give different performance, and finding the best number of clusters can be challenging.
Also, our results from the real application showed that the performance gain of our approach against the baseline increases as each constraint involves more variables, and thus, the connectivity among the variables gets stronger. Also, the proposed methodology can be easily combined with other established techniques that improve the rate of convergence, such as incremental methods (Bertsekas, 2011), smoothing techniques (Boyd et al., 2011; Nesterov, 2005), adaptive subgradient methods (Duchi, Hazan, & Singer, 2011) among others.

Last, we note that the proposed methods accelerate the convergence of the solution methods regardless of whether the sub-problems are solved in parallel or sequentially, as our experimental results show. We also note that in cases where the constraints do not exhibit any community structure (e.g., every variable appears in every constraint), the importance of variable partitioning diminishes. However, even when there is no known structure a priori, it is still possible that a structure exists, which should be identified and exploited for reducing the computation effort. A key message of this paper is that one should consider a partitioning algorithm integrating information about the structures within the problem's constraints or dependencies in the decision variables, before implementing a distributed or parallel solution method.

An extension of the proposed methodology is examining whether the proposed variable partitioning method can be applied to other parallel and distributed optimization approaches. Coordinate descent methods (Palomar & Mung, 2006; Richtárik & Takáč, 2016; Wright, 2015) have gained popularity recently. In coordinate descent methods, variables are partitioned into groups, one of which is chosen to be updated in each iteration. Thus, the approach proposed in this paper can also be used to find a partition for coordinate descent methods, which may also benefit from the community structure of decision variables found by our approach.

One limitation of the proposed approach is load balancing. None of the introduced partitioning methods guarantees sub-problems with equal sizes. A large block that dominates the execution time can affect the level of speed-up due to parallelization. From comparison to other community detection methods for block decomposition, we found that integrating knowledge about the structure in the constraint matrix or dependence in the decision variables greatly improves this aspect. One approach to complement this is to design a partitioning algorithm that penalizes the load imbalance. Another direction is to pursue an asynchronous version of the sub-gradient algorithm to reduce the impact of heterogeneous sub-problem sizes.

Acknowledgments

This research was supported by Award R01DE028283 from the National Institute of Dental and Craniofacial Research, National Institutes of Health, USA. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. The funding agreements ensured the authors independence in designing the study, interpreting the data, writing, and publishing the report. Ilbin Lee was also financially supported for this study by Xerox Faculty Fellowship from Alberta School of Business and by Discovery Grant RGPIN-2018-03960 from Natural Sciences and Engineering Research Council of Canada.

Appendix A. Numerical illustration of the general approach

The numerical results in this appendix complement those based on the transportation problem developed in the main manuscript. Specifically, we empirically illustrate the general version of our approach (Section 4.3) using multidimensional knapsack problems (MKPs).

A0.6. Multidimensional knapsack problems

An MKP is a generalization of a knapsack problem with multiple resource constraints. MKPs are used in various applications, including capital budgeting problem, project selection and cargo loading, and cutting stock problems (Chu & Beasley, 1998). In particular, large-scale instances of MKP arise in allocating computing resources to virtual machines in a distributed system (Campegianni & Presti, 2009).

A general MKP is given as follows:

\[
\text{(MKP)} \quad \min \sum_{j=1}^{n} r_j x_j \\
\text{s.t. } \sum_{j=1}^{n} W_{ij} x_j \leq b_i \text{ for } i \in I, \\
x_j \text{ binary for } j \in J,
\]

where $J$ is the set of items, and $I$ is the set of resource types. A decision variable $x_j$ indicates whether the item $j$ is chosen or not. For each item $j$ and resource type $i$, choosing item $j$ uses $W_{ij}$ units of resource $i$. For each resource type $i$, there is a constraint limiting the total amount of the resource type used. If there is only one type of resource, it becomes a standard knapsack problem. We note that choosing an item may spend only a subset of resource types, i.e., some of $W_{ij}$ may be zeros. Also, there may be a subset of variables that tend to appear together in constraints, which induces a community structure among decision variables. For example, in the problem of allocating computing resources to virtual machines, one may have to choose from a subset of physical machines due to geographical proximity to a given virtual machine. We conclude this section by noting that MKP is an integer program for which there is no strong duality for the Lagrangian formulation. The results presented in the rest of this paper are related to the LP relaxation of MKP, and its Lagrangian dual reformulation.

A0.7. Problem setup

To empirically evaluate our method, we randomly generated MKP instances with varying sizes and community structures. Let $l$, $m$, and $c$ denote the number of items, the number of resource types, and the number of communities. Given values of $l$, $m$, and $c$, we generated an MKP instance as follows. We evenly distributed items and resource types to communities. Note that in an MKP, each item corresponds to a variable, and each resource type corresponds to a constraint. We first determined which item spends which resource, i.e., which variable appears in which constraint, by constructing a bipartite graph. In the bipartite graph, nodes on one side represent items and the other side resource types. If there is an edge between $i$ and $j$, then it means that choosing item $j$ spends a positive amount of resource $j$. Starting from no edge, we added edges to the bipartite graph as follows.

We first added edges within communities by using the following procedure for each community. Note that the maximum number of edges in a community is $[lm/(c^2)]$. For each community, we randomly selected $[dlm/(c^2)]$ edges, where $d_l$ was a parameter representing the “density of connections” within a community. Each edge was chosen by randomly selecting an item and a resource type in the community. Throughout the experiments, we used 0.8 for $d_l$.

Then, we added edges between communities as follows. Note that the maximum number of possible edges between communities is $[lm – m/c]$. We randomly selected $[dm(m – m/c)]$ edges between communities, where $d_m$ is a parameter representing the “density of connections” between communities. Each edge was chosen by randomly selecting an item and a resource type that did not belong to the community to which the selected item belonged. In our experiments, we used 0.005 for the parameter $d_m$. The values of parameters $d_l$ and $d_m$ were chosen so that the resulting instances
exhibit community structure. In Section A.0.4, we performed sensitivity analysis for \( d_b \).

For each item, we randomly sampled the objective coefficient from the uniform distribution between 10 and 100. For each constraint and for each variable selected to appear in the constraint, we randomly sampled its coefficient from the uniform distribution between 50 and 100. Then, we determined the right-hand side of each constraint to be half of the sum of coefficients of the variables appearing in the constraint.

A0.8. Experiment setup

When solving the LP relaxations of the generated MKP instances, one approach we applied to partition the decision variables is the community detection method. Since MKPs do not have any spatial structure in general, we cannot compare our community detection method to the other partitioning methods, such as using prior knowledge or clustering as in Section 5.1. Instead, we generated random partitions of decision variables and obtained the average performance of the sub-gradient method for the random partitions. Regarding the number of communities of random partitioning, we used the same number used in the problem generation step. In the implementation of the sub-gradient method, we used \( \alpha_t = 100/t \) as step sizes for both the community detection and the random partitions.

For both partitioning methods, we applied the following stopping criterion for the sub-gradient method. At each iteration, we compute the current optimality gap in percentage. If the optimality gap has changed by 1% or less in the last 20 iterations, we stop the sub-gradient method. If the condition is not met by 5000 iterations, we stopped the algorithm. When the algorithm stops by satisfying the condition, we report the optimality gap at the last iteration. When it stops at 5000 iterations, we report the average optimality gap over the last 20 iterations as the algorithm might be still oscillating.

A0.9. Comparative results

Table 5 compares community detection and the random partitions for MKP instances with varying problem sizes and different numbers of communities. For each combination of parameter values (i.e., each row in the table), we generated 10 problem instances and compared the average performance. For random partitioning, we generated 30 random partitions for each problem instance and averaged the performances.

In Table 5, we first note that the optimality gap of the random partitioning is above 60% for all instances, while the community detection gives much smaller optimality gaps. Also, the number of iterations for the community detection is consistently smaller, often by a large margin, than that of the random partitioning. Fig. 6 shows the progression of the optimality gaps of the community detection and the random partitioning as a function of the number of iterations for an MKP instance with 100 items, 50 resource types, and 4 communities. The block dual decomposition with community detection not only converges faster but also leads to a much smaller optimality gap. We noted similar behavior in other instances as well. In sum, our community detection approach exhibits better convergence behavior since it utilizes the structure of the problem.

We also considered a sensitivity analysis varying the community structure. We compared the performance of the community detection and random partitions for the following instances. We generated instances with 200 items, 100 resources, and 10 communities, but using \( d_b = 0.001, 0.003, \) and 0.005. Recall that \( d_b \) is interpreted as the density of connections between communities. As \( d_b \) increases, there are more connections between communities and thus the community structure weakens. For each combination of parameters, we generated 10 random instances, and computed the average performance of the community detection over these instances. For random partitioning, we generated 30 random partitions for each instance and averaged the performance. Table 6
shows the result, and Table 7 shows the same result but for instances with 300 items, 150 resources, and 15 communities.

First, we observe that the performance of the community detection worsens as the community structure weakens, both in the optimality gap and the number of iterations. The number of iterations of the random partitioning increases as $d_{ip}$ increases, but the optimality gap either increases or does not change much. This is expected because the random partitioning does not exploit the community structure, whereas the community detection approach is designed to utilize the structure. We also tested other sizes but the implications remain the same.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2021.11.054.

References


