Toward a Parallelized Solver for Graph of Convex Sets Optimization Problems Using the Alternating Directions Method of Multipliers

Michael Zeng
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
michzeng@mit.edu

Abstract

Graphs of Convex Sets (GCS) is a recently-proposed optimization framework that solves a broad class of non-convex mixed continuous and discreet optimization problems. The implications of GCS in the field of robotics have been significant – non-convex motion-planning problems can be quickly solved by GCS with global optimality guarantees. However, as GCS has been adopted for harder and harder problems, the limitations of current computational approaches have become apparent; plugging large GCS problems into commercial solvers is no longer sufficient. In this project, I present early work toward developing a faster, parallelized solver for GCS problems using the Alternating Directions Method of Multipliers (ADMM). Specifically, we apply ADMM to the solve shortest piecewise linear paths through a graph of convex sets, and analyze the results and broader implications.

I. Class and Awknowledgements

This is a final project for the *Optimization Methods* class at MIT, so I want to make clear what work was done by myself versus others, and what was done during the time span of the class project. This project was advised by Alex Amice, a PhD student in Russ Tedrake's lab; his help was essential in teaching me the basics of GCS and ADMM, developing broad ideas of ADMM splitting methods, and checking over my detailed formulations. All work in specific problem formulation and software development were done by myself. All work described in this report was done during the span of the class project; although this constitutes a broader research project that I hope will continue beyond the class, it was started 6 weeks ago because of this class.

II. Introduction

Collision-free motion planning is a very old problem in robotics. Conventional approaches have generally been categorized into two groups–sampling-based planners, and optimization-based planners. Sampling-based planners rely on randomly sampling robot configurations and building a trajectory from those random samples. Optimization-based planners attempt to solve highly non-linear and non-convex optimization problems parameterizing the robot trajectory, subject to the robot's dynamic constraints. Both approaches face computational challenges and lack the ability to guarantee globally optimal trajectories; sampling-based planners may not sample the best trajectory, or any feasible trajectory at all; meanwhile, nonlinear optimization techniques frequently converge to local minima. However, graph of convex sets (GCS) introduces a new paradigm in motion planning. Assuming that a decomposition of the robot's collision-free space into a graph of convex set is given, the GCS solves a convex optimization and returns globally optimal trajectories at a lesser computation time than conventional methods.

GCS also has vast implications beyond simply robot motion-planning. Early work on GCS has shown applicability to a variety of problems in computer science, including the traveling salesman problem, facility location optimization, finding minimum spanning trees of graphs, and more. In the field of robotics, GCS has also shown computational advantages in problems such as footstep placement and planning for walking robots, or planning through contact [1].

However, practical implementations of GCS still face two primary challenges. Firstly, in the case of motion-planning, decomposing a robot's collision-free configuration-space into a graph of convex sets for GCS to use is a difficult problem. Moreover, the state-of-the-art for solving GCS problems is simply to plug the resulting integer programs (or, sometimes, convex relaxations of these integer programs) into commercial solvers; but if the the GCS problem contains a large or dense graph, commercial solvers struggle to solve the problem in a reasonable amount of time. This presents a barrier to applying GCS; large graphs are common, for example, in planning through challenging collision geometries or large numbers of contact modes.

Therefore, in this project, we focus on the second described challenge: developing faster computational tools to solve GCS problems. Given that this is an early work on this topic, we explore the usage of a well-known distributed optimization algorithm, the Alternating Directions Method of Multipliers (ADMM) [2]. The idea is to solve large GCS problems by taking advantage of parallel computation.

The rest of this report is structured in four sections; first: a statement of our shortest piecewise linear path problem as a GCS; second: a technical introduction to ADMM, third: technical explanation of implementations applying ADMM to the shortest piecewise linear path GCS problem; and fourth: results.

III. Problem Statement

We assume we are given a set of convex sets by the user, the union of which represents "collision-free" space that the path is allowed to be within, as well as a source point s and a target point t within these convex sets. Then, the goal of the shortest path problem is to find the minimum-length path from s to t.

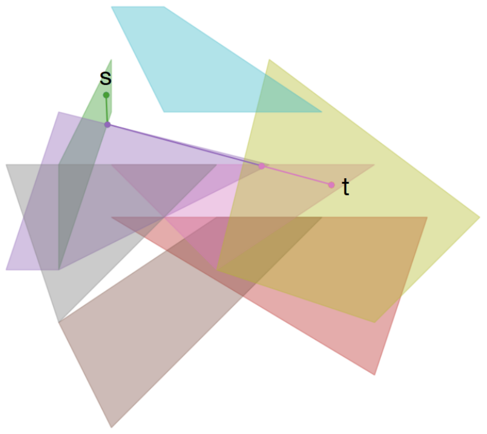

Fig. 1: The Shortest Piecewise Linear Path Problem

GCS is a good fit for this problem because the each convex set easily represents a vertex in the GCS, while edges are drawn between overlapping convex sets. The continuous positions of points on the path as well as the selection of which vertices and edges in the graph should be active in the final path are also easily modeled by GCS.

The classic GCS problem is formulated as the following mixed-integer non-convex program (MINCP) [1]:

$$\min \quad \sum_{v \in \mathcal{V}} y_v f_v(x_v) + \sum_{e=[v,w] \in \mathcal{E}} y_e f_e(x_v, x_w) \tag{1}$$

$$\text{subject to} \quad y \in \mathcal{Y} \cap \{0,1\}^{\mathcal{V} \cup \mathcal{E}}, \tag{2}$$

$$y_v x_v \in y_v X_v, \qquad \forall v \in V, \tag{3}$$

$$y_e(x_v, x_w) \in y_e X_e, \qquad \forall e = [v,w] \in E. \tag{4}$$

We model each vertex v of the graph with a continuous variable x_v and a binary variable y_v (that is 1 when v is active, and 0 otherwise), and each edge $e = [v,w]$ with a binary variable y_e. We denote the convex set at vertex v with $\mathcal{X}_v$, representing the set of valid x_v, and we define a convex set $\mathcal{X}_e$ at each edge $e = [v,w]$ that represents a valid combination of x_v and x_w.

In the shortest piecewise linear problem, we have $x_v \in \mathbb{R}^{2n}$, where n is the dimension of the space (2 in Fig. 1), representing a line segment (or, two points).

Evident in Equation (1), we encode a cost on each vertex and edge based on the values of the continuous variables x. In Equation (3) and (4), we enforce the convex set constraints only on the active vertices and edges. Finally, (2) constrains the binary variables y_v and y_e to encode valid path from s to t.

I spend the new few paragraphs of this section quickly explaining the convex reformulation of this MINCP to a mixed-integer convex program (MICP); for full details, one should see [1] ch. 5. To begin this reformulation, we specify (2) as

$$y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e + \delta_{tv}, \qquad \forall v \in \mathcal{V}, \tag{5}$$

$$\sum_{e \in \mathcal{E}_{\mathcal{U}}} y_e \leq |\mathcal{U}| - 1, \qquad \forall \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 2. \tag{6}$$

$$y_v \in \{0,1\}, \quad y_e \in \{0,1\} \qquad \forall v \in \mathcal{V}, e \in \mathcal{E} \tag{7}$$

where the scalar δ_{sv} is equal to 1 if $s = v$, and δ_{tv} is equal to 1 if $t = v$, and $\mathcal{I}_v^{\text{in}}$ and $\mathcal{I}_v^{\text{out}}$ denote the sets of incident edges flowing into and out of v, respectively. The first constraint (5) here implies flow conservation, ensuring a one unit of flow travels from the source to target; the second constraint (6) eliminates sub-tours in the presence of negative-weight cycles in the graph. We assume we have no negative-edge costs, so we drop (6) (note that dropping these constraints does weaken the convex relaxation of (24) as described below; but we do so anyway for simplicity. It is customary to retain (6) up to $|\mathcal{U}| \leq k$ where k trades tightness of the relaxation with increasing problem size) [1] [3].

The key to reformulation as a MICP is substituting out the bilinear constraints (3) and (4). To do so, we first introduce auxiliary variables, $\forall v \in \mathcal{V}, e \in \mathcal{I}_v$

$$z_v := y_v x_v \tag{8}$$
$$z_v^e := y_e x_v \tag{9}$$

These auxiliary variables allow us to transform the bilinear constraints (3) and (4) to linear constraints

$$(z_v, y_v) \in \tilde{\mathcal{X}}_v \tag{10}$$
$$(z_v^e, z_w^e, y_e) \in \tilde{\mathcal{X}}_e \tag{11}$$

where $\tilde{\mathcal{X}}_v$ and $\tilde{\mathcal{X}}_e$ are the "homogenizations" of the sets $\mathcal{X}_v$ and $\mathcal{X}_e$. This doesn't rid of the bilinearity of the problem completely, but leaves the bilinearity in constraints (16) which are much easier to reformulate:

$$\begin{aligned}
\min \quad & \sum_{v \in \mathcal{V}} y_v f_v(x_v) + \sum_{e=[v,w] \in \mathcal{E}} y_e f_e(x_v, x_w) && && (12)\\
\text{subject to} \quad & y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e + \delta_{tv} \leq 1, \quad \forall v \in \mathcal{V}, && && (13)\\
& (z_v, y_v) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V}, && (14)\\
& (z_v^e, z_w^e, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = [v,w] \in \mathcal{E}, && (15)\\
& z_v = y_v x_v, \quad z_v^e = y_e x_v, && v \in \mathcal{V},\, e \in I_v, && (16)\\
& y_v \in \{0,1\}, \quad y_e \in \{0,1\} && \forall v \in \mathcal{V}, e \in \mathcal{E}. && (17)
\end{aligned}$$

The trick to the MICP reformulation then is to replace (16) with linear constraints that "envelop" (16). Effectively, we add new constraints, called "perspective constraints" that are linear transformations of all other constraints in the problem and that "imply" (16). The exact transformation method is described in Lemma 5.1 in [1].

$$\begin{aligned}
\min \quad & \sum_{v \in \mathcal{V}} y_v f_v(x_v) + \sum_{e=[v,w] \in \mathcal{E}} y_e f_e(x_v, x_w) && && (18)\\
\text{subject to} \quad & y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e + \delta_{tv} \leq 1, && \forall v \in \mathcal{V}, && (19)\\
& z_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} z_v^e + \delta_{sv} x_v = \sum_{e \in \mathcal{I}_v^{\text{out}}} z_v^e + \delta_{tv} x_v, && \forall v \in \mathcal{V}, && (20)\\
& (z_v, y_v) \in \tilde{\mathcal{X}}_v, \quad (x_v - z_v, 1 - y_v) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V}, && (21)\\
& (z_v^e, y_e) \in \tilde{\mathcal{X}}_v, \quad (x_v - z_v^e, 1 - y_e) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V},\, e \in I_v, && (22)\\
& (z_v^e, z_w^e, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = [v,w] \in \mathcal{E}, && (23)\\
& y_v \in \{0,1\}, \quad y_e \in \{0,1\} && \forall v \in \mathcal{V}, e \in \mathcal{E}. && (24)
\end{aligned}$$

To develop a slight intuition for this, we look at (19) and (20). (20) is clearly obtained by multiplying (19) by x_v and then substituting $y_v x_v$ with z_v and $y_e x_v$ with z_v^e, which clearly is implying the bilinear relationships in (16). Doing so for all constraints in the problem results in a tight envelopment. This means this resulting MICP is an exact reformulation of the bilinear MINCP; no accuracy or information is lost.

We take this relaxation one step further by dropping (24). While this relaxation is not exact, it is, empirically, quite tight for almost all shortest path problems [1]. And [4] proposes a "rounding" step that rounds the y_v and y_e variables back to integer values by performing simple depth first searches from s to t using the non-integer values of y_e as probabilities of traversing

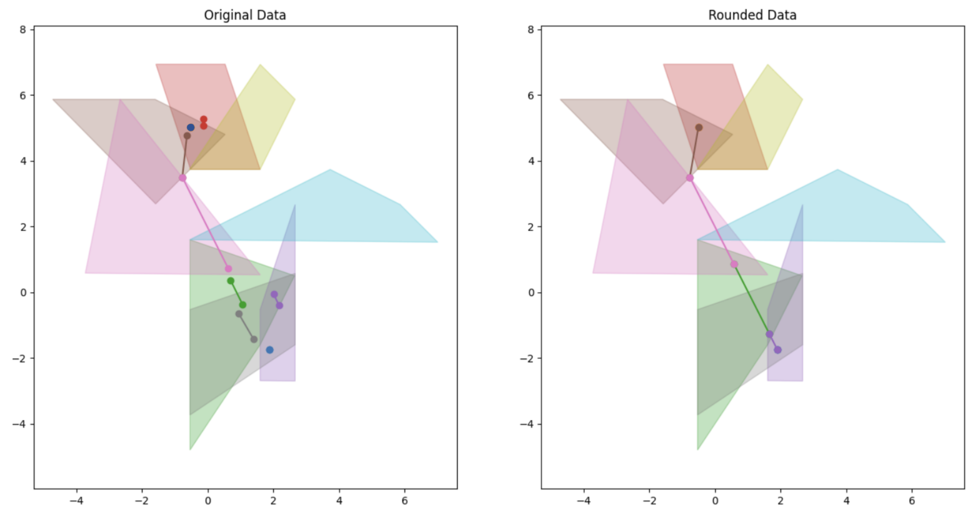

Fig. 2: Left: the unrounded solution. Right: the solution after rounding.

an edge in the graph, followed by a lightweight convex program to re-optimize the continuous variables x_v after y_v and y_e are rounded and fixed. An illustration is shown in figure 2.

The last step of our shortest piecewise linear paths GCS formulation is to fill in the specifics of the shortest piecewise linear path problem:

$$\begin{aligned}
\min \quad & \sum_{v \in \mathcal{V}} \|z_{v,1} - z_{v,2}\|_2 && && (25)\\
\text{subject to} \quad & y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e + \delta_{tv}, && \forall v \in \mathcal{V}, && (26)\\
& z_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} z_v^e + \delta_{sv} x_v = \sum_{e \in \mathcal{I}_v^{\text{out}}} z_v^e + \delta_{tv} x_v, && \forall v \in \mathcal{V}, && (27)\\
& A_v z_{v,i} \le y_v b_v, \quad A_v(x_{v,i} - z_{v,i}) \le (1 - y_v) b_v, && \forall v \in \mathcal{V},\ i = 1, 2, && (28)\\
& A_v z_{v,i}^e \le y_e b_v, \quad A_v(x_{v,i} - z_{v,i}^e) \le (1 - y_e) b_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v,\ i = 1, 2, && (29)\\
& z_{v,2}^e = z_{w,1}^e && \forall e = [v, w] \in \mathcal{E}, && (30)\\
& && && (31)
\end{aligned}$$

In this formulation, the second subscripts (i.e. $z_{v,1}$ and $z_{v,2}$) index into the continuous variables to extract the values corresponding to the 1st or 2nd point at vertex v (recall that, in our shortest piecewise linear path formulation, each vertex encodes a line segment, or two points). In addition, we assume that the convex sets describing collision-free space are given by the user as $A_v x \le b_v \ \forall v \in \mathcal{V}$. Then, (25) sums the length of each active piece of the path, (26) and (27) ensure flow conservation, (28) and (29) ensure each point at each vertex is contained in their respective convex sets, and (30) ensures that the points between connected vertices are equal to ensure a continuous path is found.

IV. ADMM

In this section, as a preface for the next section where I apply ADMM to the shortest piecewise linear paths GCS problem, I give a brief description of ADMM and how it can be applied to parallelize general convex optimization problems. Again, for further detail, see [2].

ADMM is a distributed first-order optimization algorithm applicable to any convex optimization problem of the form

$$\begin{aligned}
\min \quad & f(x) + g(z) && (32)\\
\text{subject to} \quad & Ax + Bz = c && (33)
\end{aligned}$$

with f and g convex. We will call (32) and (33) ADMM "Standard Form". Core to ADMM's functionality is the two-block form, where the variable set of the optimization problem is separable into an x and z block, whose only coupling can be described by the linear equality "consensus constraints" $Ax + Bz = c$.

The ADMM algorithm is simple. First, define the augmented Lagrangian of the ADMM problem form:

$$L_\rho(x,z,y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \tag{34}$$

Then, ADMM is an iterative algorithm that repeats these three steps until some convergence criteria are reached (shown below is the ”scaled” form of ADMM):

$$x^{k+1} := \arg\min_x \left(f(x) + (\rho/2)\|Ax + Bz^k - c + u^k\|_2^2\right), \tag{35}$$

$$z^{k+1} := \arg\min_z \left(g(z) + (\rho/2)\|Ax^{k+1} + Bz - c + u^k\|_2^2\right), \tag{36}$$

$$\mu^{k+1} := \mu^k + (Ax^{k+1} + Bz^{k+1} - c). \tag{37}$$

We call (35) the “x-minimization” step, (36) the “z-minimization step”, and (37) the “dual update” step, where μ are the “scaled” dual variables (equal to the dual variables divided by ρ). The x-minimization and z-minimization steps have the effect of moving the primal variables closer to optimal, while the dual update step moves toward satisfaction of the consensus constraints. ADMM is considered a first-order method because the dual update effectively performs “dual ascent”, updating the dual variables using the gradient (with respect to the dual variables) of the augmented Lagrangian dual function.

Lemma 1: Simply applying these iterative steps of the algorithm, one will converge to the correct answer under certain criteria: if f and g are convex, proper, and closed, and if L_ρ has a saddle point. The 2-block form is also critical to ADMM's convergence; with more or less than two blocks (i.e. if you separated the variables to more than just a x and z set), ADMM's guarantees for convergence are lost (some works have shown stricter convergence conditions in specific cases) [5] [6].

One might be wondering how it is possible to express our shortest piecewise linear path GCS problem in limited the ADMM standard form of (32) and (33). One key answer is that f and g need not be continuous or finite; they can encode hard (convex) constraints as “indictor costs”, incurring infinite costs when the constraint is violated and zero cost otherwise. Therefore, f and g can effectively be treated as constrained optimization problems during the x- and z-minimization steps.

Additionally, further conditions are needed for the algorithm to be parallelizable without losing convergence guarantees. **Lemma 2**: $f(x)$ and $g(z)$ must be further separable into $f_i(x_i)$ and $g_j(z_j)$; moreover, $x_i\ \forall i$ cannot be coupled within the consensus constraints $Ax+Bz = c$ (what this means is that there cannot be any single constraint $A_kx+B_kz = c_k$ that contains nonzero coefficients for multiple i); likewise for z_j. If these conditions are true, then the x-minimizations and z-minimizations can be divided into multiple minimizations over $f_i(x_i)\ \forall i$ and $g_j(z_j)\ \forall j$ that can be run in parallel. (To give more intuition on why the requirement that $x_i\ \forall i$ not be coupled within the consensus constraints $Ax + Bz = c$ is true, if this is not true, then the order that the coupled x_i are updated would matter, and one of these updates would have to use an outdated value (from the previous iteration of the algorithm) of the other).

An observant reader might realize that even allowing indicator costs within f and g, for most convex optimization problems including the shortest piecewise linear path GCS problem defined above, it still is not possible for reformulate in the ADMM standard form due to the strict requirements on separability. The last key to applying ADMM is the concept of creating local copies of variables. Consider, for example, if we want to separate two variables x_1 and z_1, into the x block and z block, respectively; but they are coupled by some non-linear-equality-constraints (that therefore are unable to grouped with the consensus constraints $Ax + Bz = c$). Then, we can introduce a copy of z_1 (call it z_1^x) into the x block, and a copy of x_1 (call it x_1^z) into the z block, and introduce new consensus constraints $x_1 = x_1^z$ and $z_1 = z_1^x$ to ensure consistency between all copies of the same variable. Thus, at the cost of increasing the ADMM problem size, we enable separation between the x block and z blocks without changing the solution of the optimization.

V. Applying ADMM to the Shortest Piecewise Linear Paths GCS Problem

A. *Vertex/Edge Split*

In this section, I describe the multiple ADMM splitting methods I implemented and tested. I try to do so in an intuitive manner understandable to a new user of ADMM (like myself a month ago) that describes my own learnings throughout the process.

The first step to applying ADMM to any problem is to determine how to split the x and z blocks. Recall the variable set of our shortest piecewise linear paths GCS problem:

$$\begin{aligned}
&x_v \in \mathbb{R}^{2n} && \forall v \in \mathcal{V}, \\
&z_v \in \mathbb{R}^{2n} && \forall v \in \mathcal{V}, \\
&z_v^e \in \mathbb{R}^{2n} && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v, \\
&y_v \in \{0,1\} && \forall v \in \mathcal{V}, \\
&y_e \in \{0,1\} && \forall e \in \mathcal{E}.
\end{aligned}$$

Our first idea shall be to exploit the graph structure of GCS and split the variables by vertex- and edge-ownership (we call this splitting method the "vertex-edge split"). Specifically, we define, $\forall (i,v) \in \text{enumerate}(\mathcal{V})$ and $\forall (j,e) \in \text{enumerate}(\mathcal{E})$:

$$x_i := \{x_v, z_v, y_v\}$$
$$z_j := \{z_v^e, y_e\}$$

Using these definitions of the x and z blocks, we attempt to split the piecewise linear paths GCS problem into $f_i(x_i)$, $g_j(z_j)$, and $Ax + Bz = c$:

$$f(x) \quad \begin{cases} \min & \sum_{v \in \mathcal{V}} \|z_{v,1} - z_{v,2}\|_2 \\ \text{subject to} & A_v z_{v,i} \le y_v b_v, \quad A_v(x_{v,i} - z_{v,i}) \le (1 - y_v) b_v, \qquad \forall v \in \mathcal{V},\ i = 1,2, \end{cases} \tag{38}$$

$$g(z) \quad \begin{cases} A_v z_{v,i}^e \le y_e b_v, \quad A_v(x_{v,i} - z_{v,i}^e) \le (1 - y_e) b_v, & \forall e \in \mathcal{E},\ v \in e,\ i = 1,2, \\ z_{v,2}^e = z_{w,1}^e & \forall e = [v,w] \in \mathcal{E} \end{cases} \tag{39}$$

$$Ax + Bz = c \quad \begin{cases} y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e + \delta_{tv}, & \forall v \in \mathcal{V} \\ z_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} z_v^e + \delta_{sv} x_v = \sum_{e \in \mathcal{I}_v^{\text{out}}} z_v^e + \delta_{tv} x_v, & \forall v \in \mathcal{V}. \end{cases} \tag{40}$$

where the notation $\forall v \in e$ indicates an iteration through the two vertices connected by edge e. (38) is separable into subcomponents $f_i(x_i)$ $\forall (i,v) \in \text{enumerate}(\mathcal{V})$, and (39) is separable into subcomponents $g_j(z_j)$ $\forall (j,e) \in \text{enumerate}(\mathcal{E})$. Worthy of note here is that f and g are both constrained optimizations; in fact, g has no particular cost function; they can both be thought of as indicator cost functions in the context of the ADMM standard form.

An observant reader might also notice that the second constraint in (39) actually contains variable x_v, breaking the separation of x from z; this is a major problem for the functionality of ADMM. To solve this, we have to introduce a copy of x_v $\forall v \in e$ to the each variable set z_j $\forall (j,e) \in \text{enumerate}(\mathcal{E})$. (Note that this constraint cannot be moved to the consensus constraint set because it is an inequality constraint, while the consensus constraints must be linear *equality* constraints). We will call these variable copies x_v^e. We also add a new consensus constraint to enforce consensus between x_v and its copies x_v^e $\forall v \in V, e \in \mathcal{I}_v$. The revised formulation reads:

$$f(x) \quad \begin{cases} \min & \sum_{v \in \mathcal{V}} \|z_{v,1} - z_{v,2}\|_2 \\ \text{subject to} & A_v z_{v,i} \le y_v b_v, \quad A_v(x_{v,i} - z_{v,i}) \le (1 - y_v) b_v, \qquad \forall v \in \mathcal{V},\ i = 1,2, \end{cases} \tag{41}$$

$$g(z) \quad \begin{cases} A_v z_{v,i}^e \le y_e b_v, \quad A_v(x_{v,i}^e - z_{v,i}^e) \le (1 - y_e) b_v, & \forall e \in \mathcal{E},\ v \in e,\ i = 1,2, \\ z_{v,2}^e = z_{w,1}^e & \forall e = [v,w] \in \mathcal{E} \end{cases} \tag{42}$$

$$Ax + Bz = c \quad \begin{cases} x_v = x_v^e & \forall v \in V, e \in \mathcal{I}_v, \\ y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e + \delta_{tv}, & \forall v \in \mathcal{V} \\ z_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} z_v^e + \delta_{sv} x_v = \sum_{e \in \mathcal{I}_v^{\text{out}}} z_v^e + \delta_{tv} x_v, & \forall v \in \mathcal{V}. \end{cases} \tag{43}$$

While this formulation is now properly split between x and z, rendering it a functional program if implemented in the ADMM algorithm (35) - (37), it is not yet fully parallizable. Recall from Lemma 2 that an additional requirement for parallelizability in ADMM is decoupling within the consensus constraints; this decoupling is true for the vertex variables, because each constraint in (43) only contains variables corresponding to one vertex v, but the edge variables are coupled with each other in (43) due to the summations over multiple edges. This means the x-minimizations are parallelizable with each f_i and x_i $\forall (i, v) \in$ enumerate$(\mathcal{V})$, but z-minimizations are not (and attempting to parallelize the z-minimizations leads to divergence).

This coupling among the edge variables in the consensus constraints can be handled in three ways. Firstly, we can break the coupling by introducing more copies of variables; specifically, for each z_j $\forall (j, e) \in$ enumerate$(\mathcal{E})$, we would need to introduce copies of all edge variables for all edges "adjacent" to edge e (and add new consensus constraints to enforce consensus between all the variable copies). This is a large increase in problem size and, I wouldn't consider it a good option. Secondly, $g(z)$ could be treated as one large optimization over all edge variables z without parallelization. Thirdly, the z-minimizations for each edge could be done sequentially instead of in parallel. This third option breaks the 2-block form of ADMM (instead of 2 blocks, there would be as many blocks as there are z_j $\forall (j, e) \in$ enumerate$(\mathcal{E})$, plus one for the x block), and so we lose the convergence guarantees discussed in Lemma 1, but this method surprisingly seems to converge in practice (although not very quickly).

Based on this first idea of splitting the variables of the shortest piecewise linear path GCS optimization between vertices and edges, we do arrive at two algorithms that are semi-parallelized and that do converge. These are included in our testing and evaluation.

B. *Full-Vertex Split*

Due to the limited parallelization achieved with the vertex-edge split, we now explore an entirely different splitting method we are calling the "full-vertex split", in the hopes of reaching greater parallelization. The core inspiration for this split is the observation that, in the shortest piecewise linear path GCS formulation ((25) - (30)), for each vertex, all costs and constraints only couple the variables of that vertex and of the edges incident to that vertex. Therefore, if, similarly to the vertex-edge split formulation, we separate $f(x)$ into parallel steps $f_i(x_i)$ $\forall (i, v) \in$ enumerate$(\mathcal{V})$, and each x_i is given a local a copy of the edge variables for all edges incident to v, then we can encode all constraints of the GCS optimization problem within $f(x)$ while ensuring there is no coupling between the each $f_i(x_i)$. With this idea in mind, let's begin formulating the x variable set and function $f(x)$. Specifically, we define, $\forall (i, v) \in$ enumerate$(\mathcal{V})$:

$$x_i = \{x_v, z_v, y_v\} \cup \{z_{e^v_w}, y_{e^v}\} \ \forall e \in \mathcal{I}_v, w \in e$$

We introduce some new notation here to keep things clean: $z_{e^v_w}$ denotes the copy of the continuous variable z^e_w owned by vertex v that describes the points at vertex w. Similarly, y_{e^v} describes the copy of the (relaxed binary) variable y_e owned by vertex v. As explained above, x_i effectively contains the variables of vertex v (x_v, z_v, y_v) along with a copy of every variable corresponding to the edges incident to vertex v $(z_{e^v_w}, y_{e^v}$ $\forall e \in \mathcal{I}_v, w \in e)$.

We define $f(x)$ in its separated components $f_i(x_i)$ $\forall (i, v) \in$ enumerate$(\mathcal{V})$:

$$f_i(x_i) \quad \left\{ \begin{array}{rlr} \min & \|z_{v,1} - z_{v,2}\|_2 & \\ \text{subject to} & A_v z_{v,i} \le y_v b_v, \quad A_v(x_{v,i} - z_{v,i}) \le (1 - y_v) b_v, & \forall i = 1, 2, \\ & A_v z_{e^v_v,i} \le y_{e^v} b_v, \quad A_v(x_{v,i} - z_{e^v_v,i}) \le (1 - y_{e^v}) b_v, & \forall e \in \mathcal{I}_v, \ i = 1, 2, \\ & z_{e^v_u,2} = z_{e^v_w,1} & \forall e = [u, w] \in \mathcal{I}_v \\ & y_v = \displaystyle\sum_{e \in \mathcal{I}^{\text{in}}_v} y_{e^v} + \delta_{sv} = \sum_{e \in \mathcal{I}^{\text{out}}_v} y_{e^v} + \delta_{tv}, & \\ & z_v = \displaystyle\sum_{e \in \mathcal{I}^{\text{in}}_v} z_{e^v_v} + \delta_{sv} x_v = \sum_{e \in \mathcal{I}^{\text{out}}_v} z_{e^v_v} + \delta_{tv} x_v. & \end{array} \right. \tag{44}$$

As hinted above, because $f(x)$ encompasses all costs and constraints of the shortest piecewise linear path GCS problem, $g(z)$ is empty. However, this does not mean the z variable set should be empty; as discussed in lemma 2, to ensure parallelization is possible, variables corresponding to different vertices (i.e. different x_i) must not be coupled in the consensus constraints. Therefore, to enforce consensus between the copies of edge variables for edge $e = [u, w]$ owned by vertices u and w, we cannot simply state that $y_{e^u} = y_{e^w}$, $z_{e^u_u} = z_{e^w_u}$, or $z_{e^u_w} = z_{e^w_w}$, as this would introduce obvious coupling of variables owned by u and w. Instead, we utilize the z variable set to introduce a third "edge-copy" of each edge variable; then enforce consensus of each vertex-copy with the edge-copy. Specifically, $\forall (j, e = [u, w]) \in$ enumerate$(\mathcal{E})$:

$$z_j = \{z_{e^e_u}, z_{e^e_w}, y_{e^e}\}$$

Here, the subscript-superscript e indicates ownership by the edge e. More formally then, the consensus constraints for the full vertex split formulation become:

$$Ax + Bz = c \quad \begin{cases} z_{e^e_u} = z_{e^u_u} = z_{e^w_u} \\ z_{e^e_w} = z_{e^u_w} = z_{e^w_w} \\ y_{e^e} = y_{e^u} = y_{e^w} \end{cases} \tag{45}$$

With $f(x)$, $g(z)$, and $Ax + Bz = c$ defined, if implemented in the ADMM algorithm ((35) - (37)) with parallelized x-minimizations, we would find our implementation to successfully converge.

However, there is one more step we can take to further simplify the full-vertex split formulation. As can be see in (36), with $g(z)$ empty, the z-minimization step reduces to solving for z that minimizes the augmented term of the augmented Lagrangian. However, this minimization has a trivial analytical solution: $\forall e \in \mathcal{E}$:

$$z_{e^e_u} = \frac{1}{2}(z_{e^u_u} + z_{e^w_u}) \tag{46}$$

$$z_{e^e_w} = \frac{1}{2}(z_{e^u_w} + z_{e^w_w}) \tag{47}$$

$$y_{e^e} = \frac{1}{2}(y_{e^u} + y_{e^w}) \tag{48}$$

This is intuitive; setting the edge copy of each edge variable to the average value of the vertex copies minimizes the consensus constraint residual.

Overall, with such a lightweight z-minimization step, a lightweight dual update step (due to the small number and sparse consensus constraints), and a fully-vertex-parallelized x-minimization step, we expect the full-vertex split method to achieve the best runtimes.

VI. Results

In this report so far, we've developed three solvers for our shortest piecewise linear path GCS problem; one based the vertex-edge split with a combined z-minimization, one based on the vertex-edge split with sequential z-minimization over all edges, and one based on the full-vertex split. During evaluation, we compare these three implementations to the current state-of-the-art: plugging the GCS problem directly into a commercial solver (MOSEK) [7].

To evaluate all methods, we ran each on 4 benchmark tests:

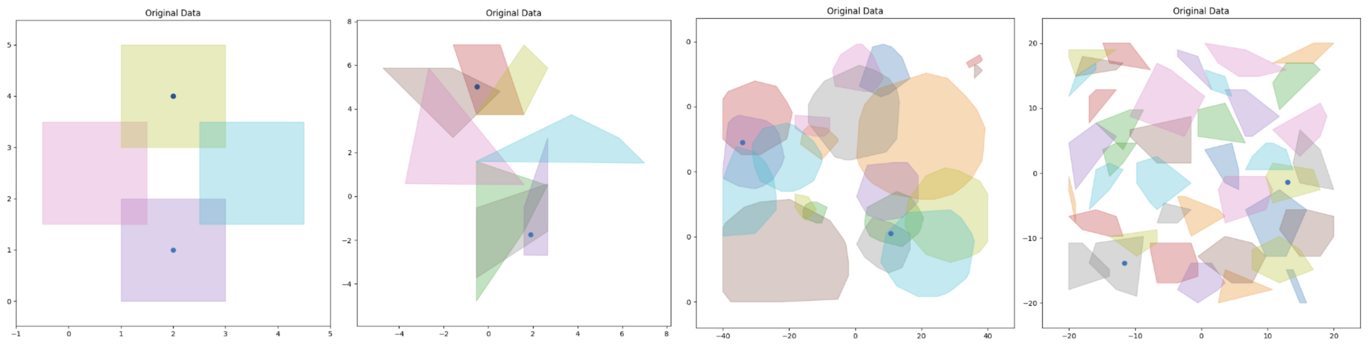

Fig. 3: Benchmark Experiments.

Due to a lack of time, I will just attach the test data. For discussion and analysis, see:

https://youtu.be/Y9mEauSYgFc?si=jlFgMoHR2gYVkax6t=1893

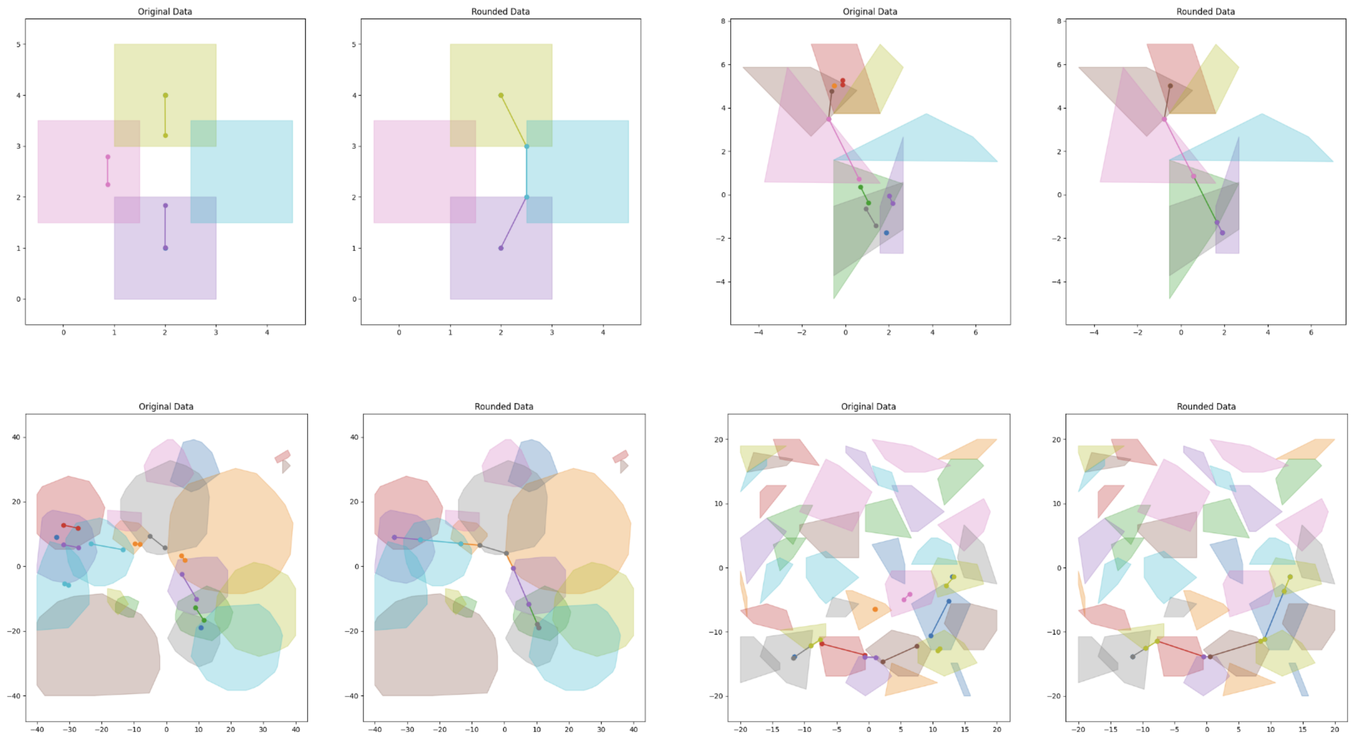

Fig. 4: Solutions from Full Vertex Split ADMM Implementation before and after Rounding Step

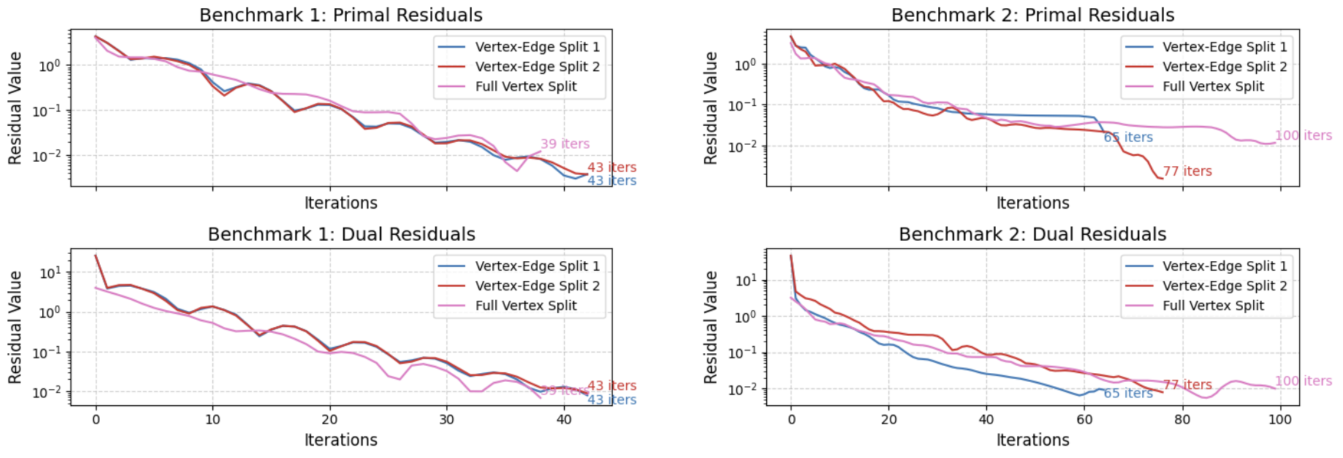

Fig. 5: Primal and Dual Residuals Plots of 3 ADMM Implementations on Benchmarks 1 and 2

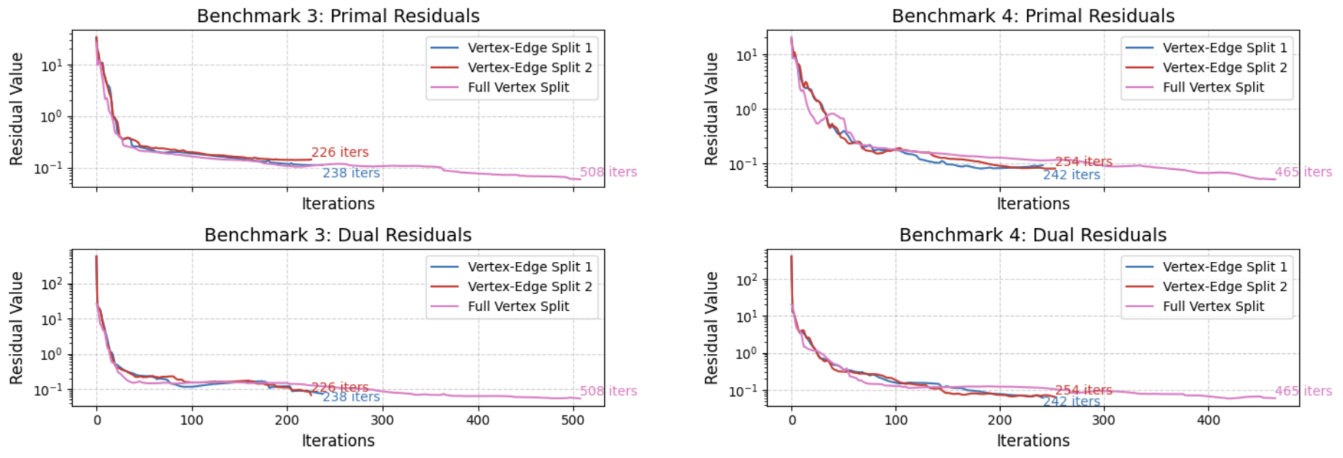

Fig. 6: Primal and Dual Residuals Plots of 3 ADMM Implementations on Benchmarks 3 and 4

Benchmark	Commercial Solver	Vertex-Edge Split 1	Vertex-Edge Split 2	Full Vertex Split
benchmark1	3.000	2.998	2.998	2.985
benchmark2	7.414	7.379	7.415	7.382
benchmark3	60.181	56.837	56.612	57.149
benchmark4	32.629	31.190	31.391	30.990

TABLE I: Cost Returned by all Algorithms on all Benchmarks

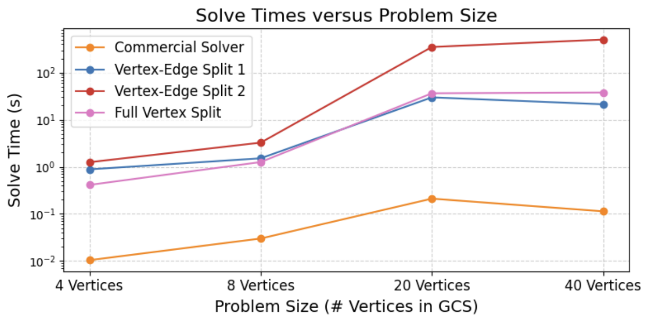

Fig. 7: Solve Times of all Algorithms on all Benchmarks

References

[1] T. Marcucci, *Graphs of Convex Sets with Applications to Optimal Control and Motion Planning*. PhD thesis, Massachusetts Institute of Technology, May 2024. Doctor of Philosophy Thesis.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. 2011.

[3] R. Tedrake and the Drake Development Team, “Drake: Model-based design and verification for robotics.” https://drake.mit.edu, 2024. Accessed: 2024-12-11.

[4] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, “Shortest paths in graphs of convex sets,” *SIAM Journal on Optimization*, vol. 34, p. 507–532, Feb. 2024.

[5] B. He and X. Yuan, “A class of admm-based algorithms for three-block separable convex programming,” *Computational Optimization and Applications*, vol. 70, no. 3, pp. 791–826, 2018.

[6] X. Cai, D. Han, and X. Yuan, “The direct extension of admm for three-block separable convex minimization models is convergent when one function is strongly convex,” *Unpublished Manuscript*, November 2014.

[7] MOSEK ApS, *The MOSEK Optimization Toolbox*, 2024. Version 10.0.