iMCB-PGO: Incremental Minimum Cycle Basis Construction and Application to Online Pose Graph Optimization

Keyu Chen, Fang Bai, Shoudong Huang, *Senior Member, IEEE*, and Yuxiang Sun, *Member, IEEE*

***Abstract*—Pose graph optimization (PGO) is a fundamental technique for robot localization. It is typically encoded with a sparse graph. The recent work on the cycle-based PGO reveals the merits of solving PGOs in the graph cycle space, which brings the computation of the minimum cycle basis (MCB) into the robotics community. However, due to batch-MCB's inability to handle the graph topology changes, it is hard for its use in real-time applications. In practice, PGOs are constructed incrementally, which requires us to solve MCB problems in an incremental setting. In this letter, we propose an exact method to solve MCB problem in an incrementally constructed graph. Methodology-wise, we first compute a tight superset called isometric set which contains an MCB, and then apply independence tests to evaporate redundant cycles to form an MCB. Our main contribution is the construction of an effective algorithm to update the superset, namely the isometric set, in an incremental setting. Our update rules preserve the optimality, thus yielding an exact incremental MCB algorithm, which is termed as iMCB. We integrate our iMCB algorithm into the cycle-based PGO, forming the iMCB-PGO approach. We validate the superior performance of our iMCB-PGO on a range of simulated and real-world datasets.**

***Index Terms*—Incremental minimum cycle basis, cycle-based pose graph optimization, SLAM back-end.**

I. Introduction

POSE graph optimization (PGO) is a prevalent approach to estimate robot poses in perception tasks such as simultaneous localization and mapping (SLAM) [1]. Recently, Bai et al. [2], [3], [4] advanced relative pose parameterizations by using all the relative poses in a graph as state variables, and constrained the cycle consistency using different cycle bases of the graph. In particular, their recent work [4], termed cycle-based PGO, see Fig. 2, emphasized the importance of using small cycles, which brings the computation of the minimum cycle basis (MCB) into the robotics community. In contrast to other relative pose parameterizations [5], [6], [7], the cycle-based PGO can be effectively solved by the sequential quadratic programming (SQP) technique. At its core, the cycle-based PGO factorizes a matrix whose structure is decided by the used cycle basis. In practice, PGO instances are rather sparse and thus have a low-dimensional cycle space. As a result, in the cycle-based PGO, the matrix to factorize has a much smaller dimension in comparison to its vertex-based nonlinear least squares (NLS) formulation [8], [9], [10], [11]. By using a minimum cycle basis, the authors [4] demonstrate the superiority of the cycle-based PGO over the classical vertex-based PGO formulation.

The cycle-based PGO requires an effective algorithm to compute an MCB, which in general is considered expensive. With that being said, for real-time SLAM applications, the pose graph is created incrementally, as new vertices and edges are added when the sensor explores the environment. This prohibits the

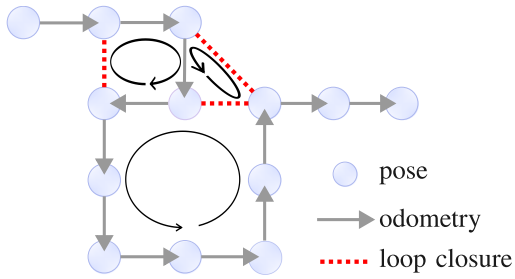
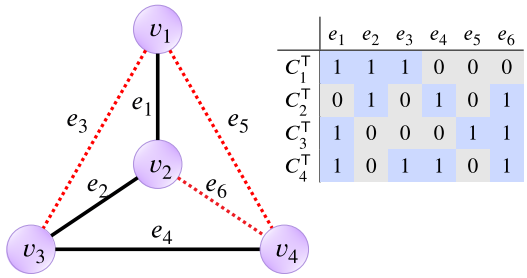


Fig. 1. Pose graph optimization in cycle space. There are three cycles corresponding to three effective constraints.



	e_1	e_2	e_3	e_4	e_5	e_6
C_1^T	1	1	1	0	0	0
C_2^T	0	1	0	1	0	1
C_3^T	1	0	0	0	1	1
C_4^T	1	0	1	1	0	1

Fig. 2. An example of the spanning tree (in black solid lines) and chords (in red dotted lines) of a graph, and indicator vectors.

Manuscript received 25 February 2024; accepted 27 June 2024. Date of publication 7 August 2024; date of current version 8 October 2024. This article was recommended for publication by Associate Editor Z. Hua and Editor C. D. Richmond upon evaluation of the reviewers' comments. This work was supported in part by Hong Kong Research Grants Council under Grant 15222523 and in part by City University of Hong Kong under Grant 9610675. *(Corresponding author: Yuxiang Sun.)*

Keyu Chen is with the Department of Mechanical Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong (e-mail: keyu.chen@connect.polyu.hk).

Fang Bai is with Surgical Augmented Reality (SURGAR), 63000 Clermont-Ferrand, France, and also with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 641651 (e-mail: fang.bai@yahoo.com).

Shoudong Huang is with the Robotics Institute, Faculty of Engineering and Information Technology, University of Technology, Ultimo, NSW 2007, Australia (e-mail: shoudong.huang@uts.edu.au).

Yuxiang Sun is with the Department of Mechanical Engineering, University of Hong Kong, Kowloon, Hong Kong (e-mail: yx.sun@cityu.edu.hk).

Digital Object Identifier 10.1109/LRA.2024.3440088

usage of the cycle-based PGO in real-time, due to its inability to handle the constant graph topology changes. Thus, it is of great importance to develop an MCB algorithm, which can incrementally handle newly added vertices and edges, of course by making the most of the obsolete MCB. We term such an MCB algorithm as an incremental MCB algorithm, *a.k.a.* iMCB in short. While an exact iMCB seems difficult, it does not compromise the research of approximate iMCB algorithms [12], for example by using the cycle constructed by a newly added edge and the shortest paths between the endpoints of the edge, an idea initialized in [13] and later matured by Forsgren et al. [14].

In this work, we propose an exact iMCB algorithm that is able to incrementally update the MCB online as new edges added. The bottleneck of the MCB algorithm [4] is the construction of the superset, which itself requires computing all-pairs-shortest-paths (APSP). Enlightened by the above fact, we report two algorithmic findings with proofs, aiming for an exact iMCB algorithm. Specifically, we propose an incremental APSP algorithm in Section III, an incremental isometric superset updating algorithm in Section IV. Both are backed with exact proofs. These two findings directly result in an efficient and exact iMCB algorithm that is integrated into the cycle-based PGO approach, forming the iMCB-PGO SLAM backend. Lastly, we integrate iMCB-PGO into proSLAM [15], a stereo-camera based SLAM system, to demonstrate the real-time performance of the proposed iMCB-PGO SLAM backend. Our code is open-sourced.[1] The contributions of this letter are summarized as follows:

1) We develop an effective incremental APSP algorithm with exact proofs. We propose the old-win-rule to handle paths with the same weight, which is much faster than the lexicographic path comparison rule in [4].
2) We develop an incremental isometric superset construction algorithm by gradually incorporating isometric cycles passing through the new edge, with proved exactness.
3) We propose an exact iMCB algorithm by incorporating the above two findings.
4) We propose iMCB-PGO by integrating the above iMCB algorithm into the cycle-based PGO.
5) We validate our results with a range of simulated and real-world datasets, including integrating iMCB-PGO into proSLAM for a real-time demonstration.

II. A Brief Review of the MCB Problem

We introduce prerequisites in graph theory, including concept of the spanning tree and chords, vector representation of cycle and concept of Horton set and isometric set.

A. Spanning Tree and Chords

In this letter, we consider a connected undirected graph $\mathrm{G}(\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the set of vertices and $\mathbf{E}$ is the set of edges. A tree is a connected subgraph without cycles or loops. A spanning tree is a special tree which contains every vertex of the graph. Given a spanning tree, the off-tree edges are called chords, as shown in Fig. 2. In general, the spanning tree of a graph is not unique.

[1]Our code is available at: https://github.com/lab-sun/iMCB-PGO

B. Cycles and GF(2)

In an undirected graph, a *cycle* is a subgraph in which every vertex has even degree. A cycle is a *circuit* if it is connected and each of its vertices has exactly degree two [16]. In this letter, the cycles we are concerned with are all circuits.

A cycle can be described by the set of traversed edges, e.g.,

$$C_1 = \{e_1, e_2, e_3\}$$

in Fig. 2. This set can be further described using an "indicator vector", where 1 means that the corresponding edge is traversed by the cycle and otherwise not. This way, the cycle C_1 can be expressed as a vector: $C_1 = \begin{bmatrix}1 & 1 & 1 & 0 & 0 & 0\end{bmatrix}^T$, as illustrated in Fig. 2. The indicator vector is a vector of ones and zeros that lies on the modulo-2 Galois field, termed GF(2) for short, where the arithmetics on GF(2) are defined by extending the standard arithmetics with the modulo-2 operation. The concept of GF(2) is useful when talking about cycles. In specific, the modulo-2 summation of two cycle vectors on GF(2) corresponds to the symmetric-difference of the corresponding two edge sets, and thus can be used to describe the cycle concatenation. The *independence of two cycles* is described by the orthogonality of the two cycle vectors on GF(2) where the inner-product of these two vectors on GF(2) equals to 0.

C. Cycles Basis

All the cycles of graph G span a *cycle space*, denoted by Cyc(G). Using the notion of GF(2), the cycle space Cyc(G) is a vector space over GF(2) spanned by all the cycle vectors, i.e., the GF(2) vector representations of cycles in graph G. A *cycle basis* of $\mathrm{Cyc}(\mathrm{G})$, denoted by B, comprises $m - n + 1$ (m is the number of edges and n is the number of vertices in G) independent cycles that can express all the cycles in G using cycle concatenations, for example via the symmetric-difference of edge sets, or via the modulo-2 summation of the corresponding cycle vectors. By independence, it is meant that any cycle in B cannot be obtained from cycle concatenations using the rest of the cycles in B. Typically, there exist multiple cycle bases in the cycle space $\mathrm{Cyc}(\mathrm{G})$, which are all equivalent under cycle concatenations. For example, in Fig. 2, a cycle basis of graph G can be chosen as $B = \{C_1, C_2, C_3\}$, where $C_1 = \{e_1, e_2, e_3\}$, $C_2 = \{e_2, e_4, e_6\}$ and $C_3 = \{e_1, e_5, e_6\}$. With this cycle basis, another cycle $C_4 = \{e_1, e_3, e_4, e_6\}$ can be expressed as $C_1 + C_2$.

D. Minimum Cycle Basis (MCB)

The *weight of a cycle* is the total weight of the edges traversed by the cycle. The *weight of a cycle basis* is the total weight of the cycles in the cycle basis. We are particularly interested in the cycle basis whose weight is minimal, termed a *minimum cycle basis* (MCB) [16]. If each edge in the graph has weight 1, an MCB is also a minimum length cycle basis (MLCB). In this work, we are concerned with the computation of an MCB, by assuming the edge weights are positive.

There exist a body of literature studying the MCB problem, see [16] for a review. In early days, people held a belief that an MCB problem was NP-hard. The first breakthrough comes from Horton [17], who identified a superset of MCBs, which brings the computation of an MCB into polynomial time. In his work, Horton found that for a cycle in an MCB, any two of its vertices are connected by a shortest path in the cycle. Thus, the potential

candidate cycle C of MCB can be represented by an edge-vertex pair (v_x, e_{uw}), which is called a Horton cycle:

$$C(v_x, e_{uw}) \stackrel{\text{def}}{=} P_{xu} \cup e_{uw} \cup P_{xw}, \tag{1}$$

where P_{ab} is a shortest paths between vertex a and vertex b. The set of Horton cycles, called Horton set, is a superset of an MCB, defined as

$$H = \{C(v_x, e_{uw}) | v_x \in \mathbf{V}, e_{uw} \in \mathbf{E}\}. \tag{2}$$

Lastly, an MCB can be extracted by applying an independence test, e.g., using Gaussian elimination on GF(2) as in [17].

Orthogonally, DePina [18] made another important breakthrough by discovering the concept of support vector. A set of support vectors, incrementally constructed with an update law, form the orthogonal complementary space to an MCB on GF(2). In [18], DePina showed how an MCB can be constructed progressively by finding a Horton cycle each time that is non-orthogonal to the current support vector. DePina's support vectors fundamentally reshaped people's understanding on cycle independence. Practically, to benefit faster computation, independence tests are restricted to off-tree edges, by removing the edges of a spanning tree in Horton cycles and support vectors. In later work from Amaldi et al. [19], a super fast independence test algorithm, such a spanning tree is constructed incrementally.

For an implementation of Horton and DePina's approaches, we refer to [20].

E. Recent Advancement

The Horton set contains many redundant cycles. In particular, Horton found that if the shortest paths are unique, each Horton cycle (of N vertices) will contain exactly N duplicates. If there exist multiple shortest-paths, the same property can be obtained by using the idea of consistent shortest paths.

Definition 1: Consistent shortest paths. Let P_{xy} denote the shortest path between vertices x and y. For any two vertices s and t on P_{xy}, if P_{st} is contained by P_{xy}, i.e., $P_{st} \subseteq P_{xy}$, the path P_{xy} is called consistent shortest path.

Definition 2: Isometric cycles. A cycle C is isometric if for any two vertices x and y on C, P_{xy} is contained by C, i.e., $P_{xy} \subseteq C$.

Amaldi [21] systematically studied the property of the isometric cycles, and found that given consistent shortest paths, the set of isometric cycles, called isometric set, contains an MCB.

Lemma 1. ([21]): The isometric set, a.k.a., the set of isometric cycles constructed from consistent shortest paths, contains a minimum cycle basis.

In addition, Amaldi et al. [21] established a method to extract the isometric set from the Horton set, by assuming the shortest paths are consistent. The key idea is to identify all the equivalent cycles in the Horton set, by a result recapitulated as follows:

Lemma 2. ([21]): Let $s_x(y)$ be the first vertex (except x) on the shortest path P_{xy}. For any cycle $C = C(x, e(u, v))$, with $e(u, v) \notin P_{xu}$, $e(u, v) \notin P_{xv}$, and $s_x(u) \neq s_x(v)$.

1) *If $x = u$ then $C = C(v, e(u, v))$.*
2) *If $x \neq u$, let $x' = s_x(u)$.*
a) *If $x = s_{x'}(v)$ then $C = C(x', e(u, v))$.*
b) *If $x \neq s_{x'}(v)$, $u = s_v(x')$ then $C = C(v, e(x, x'))$.*
c) *If $x \neq s_{x'}(v)$, $u \neq s_v(x')$ then C is not isometric.*

The cycle equivalence can be summarized as a graph $\mathrm{G}^\dagger$, where each vertex in $\mathrm{G}^\dagger$ denotes a Horton cycle, and an oriented edge $h_1 \to h_2$ $(h_1, h_2 \in H)$ describes the equivalence obtained from h_1 to h_2 using Lemma 1. In [21], Amaldi proved that equivalent Horton cycles form connected components in $\mathrm{G}^\dagger$, and furthermore an isometric cycle (of N vertices) corresponds to a connected component in $\mathrm{G}^\dagger$ with exactly N vertices. Later, Bai et al. [4] proved that the connected component is actually a double-linked circuit, which enables compact storage and fast traversal on $\mathrm{G}^\dagger$, using a C++ vector-based implementation.

Lemma 3. ([4]): All representations of an isometric circuit belong to the same connected component.

Lemma 4. ([4]). All representations of an isometric circuit C in $\mathrm{G}^\dagger$ form a double-linked directed cycle with $|C|$ vertices.

The isometric set is a tight superset to MCB, with few redundancies, from which an MCB can be effectively extracted using independence test. We use Amaldi's [19] independence test algorithm which has been implemented in [4].

The remaining issue, and of course the computational bottleneck of MCB, is the construction of consistent shortest paths. In [4], Bai et al. approached this problem by a modified Dijkstra algorithm, termed LexDijkstra, which replaces the path comparison in Dijkstra using the lexicographic comparison.

Definition 3: Lexicographical Comparison Rule (LCR). There exists a unique path P_{xy} that satisfies exactly one of the following three conditions with respect to any other path P'_{xy}.

1) $\omega(P_{xy}) < \omega(P'_{xy})$
2) $\omega(P_{xy}) = \omega(P'_{xy})$, $\text{len}(P_{xy}) < \text{len}(P'_{xy})$
3) $\omega(P_{xy}) = \omega(P'_{xy})$, $\text{len}(P_{xy}) = \text{len}(P'_{xy})$ and $\text{min\_index}(P_{xy} \setminus P'_{xy}) < \text{min\_index}(P'_{xy} \setminus P_{xy})$.

$\text{min\_index}(P)$ is defined as the minimal edge index on path P.

The lexicographic comparison ensures the uniqueness and consistency of shortest paths [22]. The most expensive part in lexicographic comparison is the last step, which requires path traversals. However, it suffices to compare to a common shared vertex, rather than the root of the shortest path tree.

Lemma 5. ([4]): In lexicographic comparison, the path traversals in the worst case stop at a common vertex shared by the two paths.

However, the LexDijkstra algorithm in [4] does not scale well for graphs created incrementally, as which may result in redundant computations if the shortest paths between two vertices are known and unchanged. Therefore, we require a new method to update consistent shortest paths incrementally, and this letter bridges this gap by proposing a set of update rules with proved exactness.

III. Incrementally Updating Shortest Paths

Given the graph with V vertices and E edges, all-pair shortest paths are stored in the form of all-pair shortest path trees (APSPT), a matrix each entry of which is the pair of distance information and path information. Distance information contains weight ω and length l, and path information is represented by predecessor vertex Pre. For example, $Pre(x, y)$ is the vertex preceding y along the shortest path from x to y, with distance ω_{xy} and length l_{xy}, as shown in Fig. 4(a).

A. Incremental Reduction

There are two cases when adding a new edge, forming a bridge and forming a cycle. The vertex from which the trajectory starts in the current loop is caller v_a. A protruded vertex, represented as v_u, is a newly introduced vertex in our evolving graph, stemming

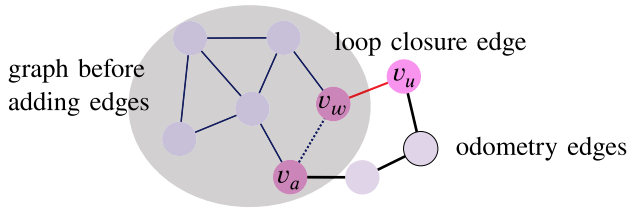


Fig. 3. Topology of pose graph growing. The pose graph extends from v_a to v_u over odometry edges (black) and loop closure edge (red). The dashed line represents the connection between v_a and v_w, which is omitted for clarity.

from the trajectory's extension and is anticipated to participate in an upcoming loop closure. Another vertex in the loop closure is named as v_w. Fig. 3 shows the topology of graph growing.

In a path update process, we are often required to compare two paths (e.g., an old path and a new path). If one path is strictly shorter than the other in distance, we simply choose the shorter one. However, things become complicated if two paths have exactly the same distance. The LCR, defined in Definition 3, is one approach to disambiguate two paths in this case, by further comparing lengths and edge IDs of non-intersected parts. Here, we notice in an incremental setting, it suffices to use an *Old-Win-Rule (OWR)* in most cases.

Definition 4: Old-Win-Rule (OWR): Let P_{xy} be the old $x-y$ path and P'_{xy} be the new $x-y$ path (passing through the added new loop-closure edge). If $d(P_{xy}) = d(P'_{xy})$, we choose P_{xy} as the shortest path between x and y.

The OWR in Definition 4 takes advantage of the fact that the shortest paths in G are already consistent, which greatly reduces the usage of LCRs and thus results in more effective updating processes.

B. Updating Shortest Paths When Adding a Loop-Closure Edge

In this case, the added new edge is e_{uw}, which introduces a new cycle in graph G′, see Fig. 3 and Fig. 4(d). To evaluate the impact of e_{uw}, we evaluate all pairs shortest paths in G′, and update those that are subject to changes.

For any pair of vertices v_s, v_d where $v_s, v_d \neq v_u, v_w$, the insertion of edge e_{uw} contributes two additional paths between v_s and v_d:

$$\begin{cases} P_1 = P_{su} \cup e_{uw} \cup P_{wd} \\ \omega_1 = \omega_{su} + \omega'_{uw} + \omega_{wd} \\ l_1 = l_{su} + l'_{uw} + l_{wd} \end{cases} \text{and} \quad \begin{cases} P_2 = P_{sw} \cup e_{uw} \cup P_{ud} \\ \omega_2 = \omega_{sw} + \omega'_{uw} + \omega_{ud} \\ l_2 = l_{sw} + l'_{uw} + l_{ud} \end{cases}$$

Degenerate cases where v_s or $v_d = v_u$ or v_w induce degenerate paths $P_{uu} = \varnothing$ or $P_{ww} = \varnothing$. We compute the shortest path between s and d in G′, denoted by P'_{sd}, in three cases.

Case 1: If $\omega_1 < \omega_{sd}$ or $\omega_2 < \omega_{sd}$ and $\omega_1 \neq \omega_2$, we choose the shorter path as the new shortest path between v_s and v_d. The pointers are updated as:

$$\omega'_{sd} = \omega_1, l'_{sd} = l_1, Pre'(v_s, v_d) = Pre(v_w, v_d) \quad \text{if } \omega_1 < \omega_2$$

$$\omega'_{sd} = \omega_2, l'_{sd} = l_2, Pre'(v_s, v_d) = Pre(v_u, v_d) \quad \text{if } \omega_1 > \omega_2$$

Case 2: If $\omega_1 = \omega_2 < \omega_{sd}$, we disambiguate P_1 and P_2 by using the LCR to choose the lexicographically shorter path. The corresponding pointers are updated similarly.

Case 3: Otherwise, i.e., $\omega_1, \omega_2 \geq \omega_{sd}$, we stick to the old path by the OWR.

We assume the shortest paths in graph G are consistent. In what follows, we show that the shortest paths in the new graph G′ are also consistent following the above rules.

Theorem 1: For any two vertices $v_s, v_d \in \mathbf{V}$ with v_s, $v_d \neq v_u, v_w$ and two arbitrary vertices v_p and v_q on path P'_{sd}, we have $P'_{pq} \subseteq P'_{sd}$.

Proof: We prove Theorem 1 by evaluating the weights of two new paths P_1 and P_2 in two cases: 1) P_1 or P_2 and 2) old path.

No matter which path we choose (P_1 or P_2 and old path), we prove that for any two vertices v_p and v_q on the chosen path, the subpath ended by v_p and v_q is shortest path.

See Appendix for details of the two cases ■.

C. Updating Shortest Paths When Adding an Odometry Edge

An odometry edge which connects an existing vertex v_a in G and a newly added vertex v_u in G′, does not change the cyclic structure of the graph. Therefore, the shortest path between any vertex v_s in G to v_u, denoted by P'_{su} in G′, can be trivially obtained by connecting the shortest path P_{sa} and the new edge e_{au}, that is $P'_{su} = P_{sa} \cup e_{au}$. The corresponding pointers are updated as follows.

For any vertex $v_s \neq v_u, v_a$,

$$\begin{aligned} Pre'(v_u, v_s) = Pre(v_a, v_s), Pre'(v_u, v_a) = v_u \\ l'_{us} = l_{as} + l_{ua}, \quad \omega'_{us} = \omega_{as} + \omega_{ua} \end{aligned} \tag{3}$$

D. Smoothing Out Vertices of Degree Two for Faster Computation

A chain of vertices of degree two (VoDT), e.g., a sequence of odometry edges without a loop-closure, does not impact the cyclic structure of a graph. However, these VoDT can increase the complexity of shortest path search, and cause extra expenses on managing the APSPT. To mitigate this issue, Bai et al. [4] eliminated the chain of VoDT, for instance from v_p to v_q, by replacing it with a crafted edge e_{pq}.

In an incremental algorithm, for a loop-closure edge, it is possible that one of its end vertices, say v_w, lies on the VoDT chain. If so, v_w should be marked as non-VoDT (activated) first, as shown in Fig. 4(c). This step will incur the computation of shortest paths between all vertices and v_w without changing the distance between any previous pair of vertices. The intermediate vertex v_w could only be reached through its neighbouring vertices, say v_x and v_y. Thus the shortest path from any vertex v_s to the intermediate vertex v_w is determined by the shortest paths from v_s to v_x and v_y. The pointers are updated as follows.

For any vertex v_s, if $\omega_{sx} + \omega_{xw} < \omega_{sy} + \omega_{yw}$,

$$\begin{aligned} Pre'(v_s, v_w) = v_x \\ l'_{sw} = l_{sx} + 1, \omega'_{sw} = \omega_{sx} + \omega_{xw} \end{aligned} \tag{4}$$

otherwise,

$$\begin{aligned} Pre'(v_s, v_w) = v_y \\ l'_{sw} = l_{sy} + 1, \omega'_{sw} = \omega_{sy} + \omega_{yw} \end{aligned} \tag{5}$$

It is noteworthy that the length of all shortest paths that go through the path P_{xy} increases by 1.

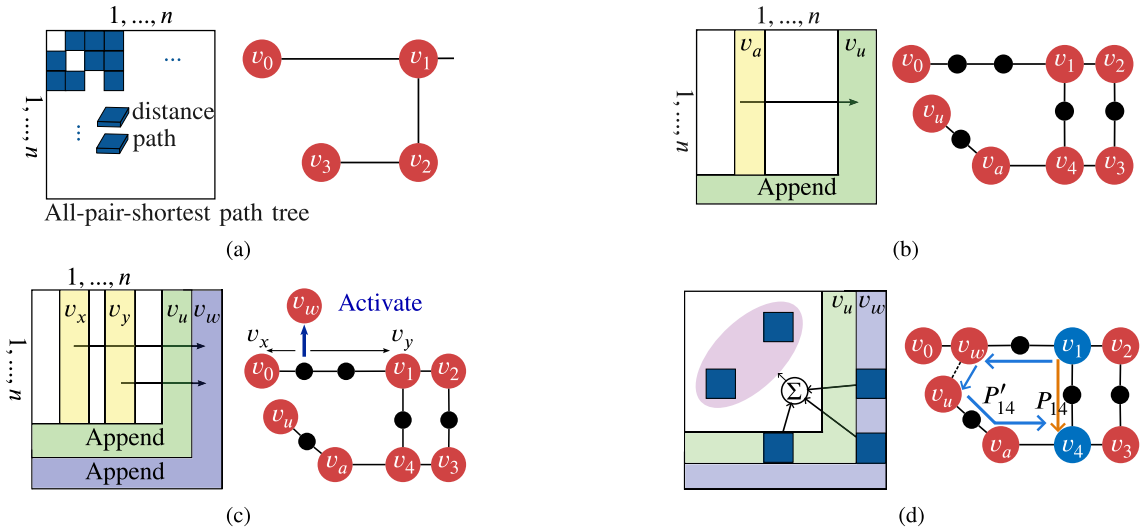


Fig. 4. Illustration of building reduced graph incrementally and performing update of all-pair-shortest paths. Original and reduced graph are both presented in (a)–(d), where red vertices are active vertices, representing nodes in reduced graph while black nodes only represent ordinary nodes in original graph. (a) Shortest paths are stored in a matrix whose entry $\mathrm{M}(x, y)$ includes $Pre(x, y)$ and ω_{xy}. (b) Trajectory extends from vertex v_a to vertex v_u, which is the additional active vertex as a potential vertex of degree two due to upcoming loop closure with v_w. (c) If v_w is an ordinary vertex of original graph, it is activated first and then its two neighbors v_x and v_y are determined by searching both forward and backward. (d) Insert new edge and update all-pair-shortest paths. The pair of two blue nodes selected randomly to illustrate how the path is renewed.

IV. Incrementally Updating the Isometric Superset

A. The Superset $\mathcal{S}$

We denote the isometric set in the previous graph G by $\mathcal{I}_{\mathrm{G}}$, and that in the current graph G′ by $\mathcal{I}_{\mathrm{G}'}$. We denote the set of new isometric cycles in G′ passing through new edge e_{uw} by:

$$\mathcal{I}_{pass\_e} \overset{\text{def}}{=} \{C | C \in \mathcal{I}_{\mathrm{G}'} \text{ and } e_{uw} \subseteq C\}. \tag{6}$$

Importantly, we note that after updating the APSP, some isometric cycles in G become non-isometric in G′. We opt to keep those non-isometric cycles in the superset construction. In G, we denote such a superset as $\mathcal{S}_{\mathrm{G}}$ which includes $\mathcal{I}_{\mathrm{G}}$, and some non-isometric cycles derived from previous rounds of APSP changes. In G′, some isometric cycles in $\mathcal{I}_{\mathrm{G}}$ (and thus $\mathcal{S}_{\mathrm{G}}$) becomes non-isometric. However, we show that the union of $\mathcal{S}_{\mathrm{G}}$ and $\mathcal{I}_{pass\_e}$:

$$\mathcal{S}_{\mathrm{G}'} = \mathcal{S}_{\mathrm{G}} \cup \mathcal{I}_{pass\_e} \tag{7}$$

contains the isometric set $\mathcal{I}_{\mathrm{G}'}$, and thus an MCB in G′.

Theorem 2: $\mathcal{S}_{\mathrm{G}'}$ contains an MCB of the updated graph G′.

Proof: We prove $\mathcal{S}_{\mathrm{G}'}$ contains an MCB by proving the isometric set of G′, is a subset of $\mathcal{S}_{\mathrm{G}'}$, i.e., $\mathcal{I}_{\mathrm{G}'} \subseteq \mathcal{S}_{\mathrm{G}'}$.

We partition the cycles in $\mathcal{I}_{\mathrm{G}'}$ into two disjoint subsets

$$\mathcal{I}_{\mathrm{G}'} = \mathcal{I}_{\mathrm{G}'_1} \cup \mathcal{I}_{\mathrm{G}'_2}, \tag{8}$$

where $\mathcal{I}_{\mathrm{G}'_1}$ is the set of all unchanged isometric cycles from G, i.e., $\mathcal{I}_{\mathrm{G}'_1} \subseteq \mathcal{I}_{\mathrm{G}}$, see Fig. 5. Then we prove $\mathcal{I}_{\mathrm{G}'_2} = \mathcal{I}_{pass\_e}$ by proving $\mathcal{I}_{\mathrm{G}'_2} \subseteq \mathcal{I}_{pass\_e}$ and $\mathcal{I}_{pass\_e} \subseteq \mathcal{I}_{\mathrm{G}'_2}$.

Assume there exists a cycle $C_i \in \mathcal{I}_{\mathrm{G}'_2}$ that does not pass through edge e_{uw}, i.e., $e_{uw} \notin C_i$. In this case, by OWR, any subpath $P'_{xy} \subseteq C_i$ sticks to the old path, which means $P_{xy} = P'_{xy} \subseteq C_i$. The above fact in essence asserts that $C_i \in \mathcal{I}_{\mathrm{G}'_1}$, which contradicts $C_i \in \mathcal{I}_{\mathrm{G}'_2}$. Thus we have $\mathcal{I}_{\mathrm{G}'_2} \subseteq \mathcal{I}_{pass\_e}$. Lastly, since $\mathcal{I}_{pass\_e} \subseteq \mathcal{I}_{\mathrm{G}'}$ but $\mathcal{I}_{pass\_e} \cap \mathcal{I}_{\mathrm{G}'_1} = \varnothing$, we know $\mathcal{I}_{pass\_e} \subseteq \mathcal{I}_{\mathrm{G}'_2}$.

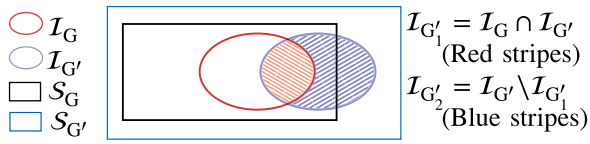


Fig. 5. Illustration for set relationship in Theorem 2.

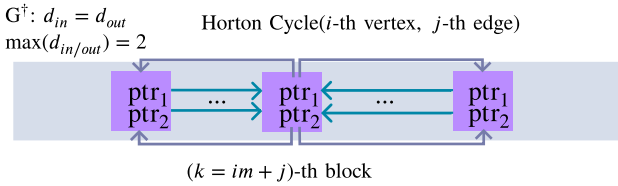


Fig. 6. The vector form storage of G†. A Horton cycle constructed from the i-th vertex and the j-th edge is mapped to position $im + j$ in the vector, with m being the number of edges in G. Each vector slot maintains two pointers ptr_1 and ptr_2 that provide links to equivalent Horton cycles.

With $\mathcal{I}_{\mathrm{G}'_1} \subseteq \mathcal{I}_{\mathrm{G}}$ and by definition of $\mathcal{S}_{\mathrm{G}'}$, we have $\mathcal{I}_{\mathrm{G}'} \subseteq \mathcal{S}_{\mathrm{G}'}$ ■.

B. Details of Constructing $\mathcal{I}_{pass\_e}$

Initialized from the empty set, the superset $\mathcal{S}$ is updated incrementally, by including isometric cycles passing through the new edge each time. The isometric cycles are obtained by identifying equivalent Horton cycles using a graph structure, denoted by G†. Each Horton cycle is a vertex in G†, and two equivalent Horton cycles C_1 and C_2 contribute two directed edges $C_1 \to C_2$ and $C_2 \to C_1$ from Lemma 2. Graph G† is huge, but the connected components in G† are double linked circuits [4]. Thus, we can store G† compactly in a vector, where each vertex of G† corresponds to a vector slot which maintains two pointers to describe edge adjacency, see Fig. 6.

TABLE I
COMPLEXITY OF IMCB AND BATCH-MCB

Method	Complexity		
iMCB	Update APSP	Regenerate graph	Update superset
	$O(n^2)$	$O(mn)$	$O(m)$
batch-MCB	Compute APSP	Generate graph	Build superset
	$O(nm(logn+n))$	$O(mn)$	$O(mn)$

n is the number of vertices, and m is the number of edges.

TABLE II
TOTAL WEIGHTS OF MCBS COMPUTED BY IMCB AND BATCH-MCB ON CLASSICAL DEBUG GRAPHS

	Folkman	Heawood	Petersen	Kneser		
				$K(5,1)$	$K(6,2)$	$K(7,3)$
iMCB	57	48	30	18	109	217
batch-MCB	57	48	30	18	109	217
		n-cube		Circulant		
	4-cube	7-cube	10-cube	C(5,2)	C(6,1,2,3)	C(10,2,4)
iMCB	68	1284	16388	15	30	60
batch-MCB	68	1284	16388	15	30	60

After updating APSP, we recompute the elements of this vector to accommodate the possible changes of cycle equivalences.

Lemma 6 ([21]): For an isometric cycle C, there is a unique edge e for each vertex $v_x \in C$, such that $C = C(v_x, e)$.

Based on Lemma 6, $\mathcal{I}_{pass\_e}$ can be obtained by traversing graph $\mathrm{G}^{\dagger}$ via two depth-first searches (DFSs) initialized from vertices $C(v_u, e)$ and $C(v_w, e)$, where $e = e_{uw}$ is the new added edge. The amortized complexity of these two DFSs is $O(m)$.

C. Overall Complexity

We extract an MCB from the superset $\mathcal{S}$ using the Adaptive Isometric Cycles Extraction (AICE) algorithm in Amaldi et al. [19], which is very fast after sorting all isometric cycles in $O(m \log m)$. Updating shortest paths using the new edge takes $O(n^2)$ time as each pair is checked. In the construction of superset $\mathcal{S}$, it takes $O(mn)$ time to regenerate graph $\bar{\mathrm{G}}^{\dagger}$ and takes $O(m)$ time to update the superset. Overall, the complexity of our iMCB algorithm is $O(n^2 + mn)$, see Table I.

Compared to the batch MCB algorithm in Bai et al. [4] with a complexity $O(nm(logn+n))$ of batch MCB algorithm, the key progress of iMCB is twofold. First, compared to general Dijkstra algorithm, we do not need to traverse each edge, and thus we break mn^2 by solely checking if a new path passing through immediate vertex is shorter between all pairs. Second, by using OWR, we substantially reduce the usage of the expensive LCR for comparing two paths.

V. EXPERIMENTAL RESULTS AND DISCUSSIONS

We evaluate the proposed iMCB and its application in PGO termed iMCB-PGO for real-time SLAM systems.

A. Test on Classical Graph

We compute the total weight of each computed MCB on classical debug graphs, that is, Folkman, Heawood, Petersen, Kneser, Hypercube and Circulant graphs, using iMCB and batch-MCB algorithms and report the results in Table II. For each case, the total weight of MCB computed by iMCB is exactly the same as that computed by batch-MCB, which backs the correctness of our incremental updating strategies.

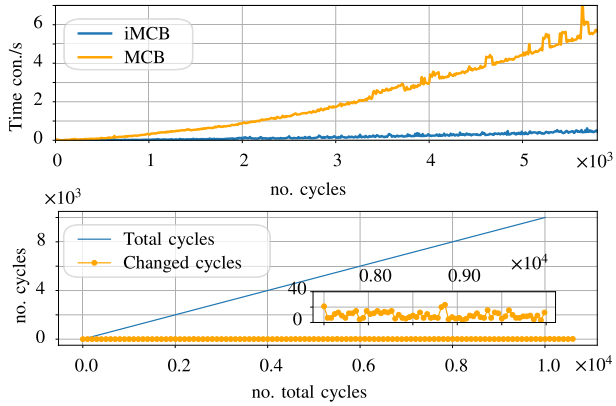


Fig. 7. iMCB on City10 k. Top: Runtime of iMCB over the number of cycles. Bottom: The number of changed cycles.

B. Time Comparison of MCB and iMCB

We evaluate the computational complexity of our iMCB over the batch-MCB in [4] on standard PGO benchmarks, and report timing statistics (in seconds) in Table III. For each case, the weights of the MCBs computed from iMCB and batch MCB are identical and thus not reported any more. In Table III, we report the detailed timing statistics of each algorithmic component to process the very last loop-closure, and the average overall timing statistics to process each loop-closure.

This result shows that the processing time spent on each loop-closure is significantly reduced by using iMCB over batch-MCB, in particular for large and dense graph instances, for example M3500, Sphere2500 and City10 K. The statistics of the last loop-closure, with iMCB being several times faster than batch-MCB, validates the effectiveness of the incremental updating strategy thanks to its capability to take advantage of existing computation to process new loop-closures for faster running time.

Fig. 7 shows the results on City10 k, a high cycle-ratio and large-scale pose graph. We report the time used to process each loop-closure edge by iMCB and batch-MCB respectively, see Fig. 7(a). Asymptotically, iMCB scales much better than batch-MCB with respect to growing graphs, because iMCB reuses existing computation (most importantly in APSP) to process a newly added loop-closure edge rather than computes everything from scratch. Such a property is critical to handle incrementally constructed PGO instances online. Fig. 7(b) illustrates that the cycles requiring replacement exhibit a nearly constant number throughout the graph's growth.

C. iMCB-PGO and Classical Vertex-Based PGO

We compare iMCB-PGO with the classical vertex-based PGO (VB-PGO). To show the point, we simulate a sparse sphere by deleting loop closure edges randomly from g2o_sphere, and report in Table IV the average error and runtime. It is clear that iMCB-PGO can converge to the final accuracy mostly in one iteration with less time consumption than that of VB-PGO. In addition, the time used for updating MCB is lower

TABLE III
COMPUTATIONAL TIME (IN SECONDS) BY IMCB AND BATCH MCB

Dataset	no. vertices	no. edges	Total (last loop)		APSP (last loop)		Isometric set (last loop)		Average time/loop	
			iMCB	MCB	iMCB	MCB	iMCB	MCB	iMCB	MCB
MITb	808	827	$\mathbf{1.13 \times 10^{-4}}$	1.72×10^{-4}	$\mathbf{1.70 \times 10^{-5}}$	4.65×10^{-5}	$\mathbf{1.49 \times 10^{-5}}$	3.23×10^{-5}	$\mathbf{6.18 \times 10^{-5}}$	9.47×10^{-4}
INTEL	1228	1483	2.06×10^{-3}	1.49×10^{-3}	9.13×10^{-4}	6.49×10^{-4}	8.14×10^{-4}	4.60×10^{-4}	$\mathbf{1.36 \times 10^{-3}}$	2.01×10^{-3}
M3500	3500	5453	$\mathbf{1.68 \times 10^{-1}}$	3.10×10^{-1}	$\mathbf{1.32 \times 10^{-1}}$	2.10×10^{-1}	$\mathbf{3.21 \times 10^{-2}}$	1.95×10^{-1}	$\mathbf{3.48 \times 10^{-2}}$	1.04×10^{-1}
Sphere2500	2500	4949	$\mathbf{9.57 \times 10^{-2}}$	7.10×10^{-1}	$\mathbf{3.10 \times 10^{-2}}$	6.20×10^{-1}	$\mathbf{3.70 \times 10^{-2}}$	8.51×10^{-2}	$\mathbf{2.28 \times 10^{-2}}$	2.10×10^{-1}
CIty10k	10000	20687	**1.11**	4.78	$\mathbf{7.85 \times 10^{-1}}$	3.75	$\mathbf{2.25 \times 10^{-1}}$	1.00	$\mathbf{5.51 \times 10^{-1}}$	1.74

TABLE IV
COMPARISON FOR A SPARSE SPHERE2500 (3291 EDGES, 2500 VERTICES AND 792 CYCLES)

Iterations	Method	Average error		objective function	Time(s)	iMCB+Opt (s)	Average time (s) for processing each loop closure
		Rotation (deg/100m)	Translation (%)				
N=1	iMCB-PGO	$\mathbf{9.40 \times 10^{-3}}$	**1.82**	$\mathbf{1.74 \times 10^{3}}$	**263.5**	20.4+243.0	**0.33**
	VB-PGO	1.70×10^{-1}	54.20	7.59×10^{7}	285.2		0.36
N=2	iMCB-PGO	$\mathbf{9.50 \times 10^{-3}}$	**1.83**	$\mathbf{1.74 \times 10^{3}}$	**413.0**	20.3+392.6	**0.52**
	VB-PGO	1.72×10^{-2}	3.65	1.39×10^{5}	537.2		0.68
N=3	iMCB-PGO	9.50×10^{-3}	1.82	1.74×10^{3}	**494.5**	20.4+474.2	**0.62**
	VB-PGO	9.50×10^{-3}	1.82	1.74×10^{3}	776.0		0.98

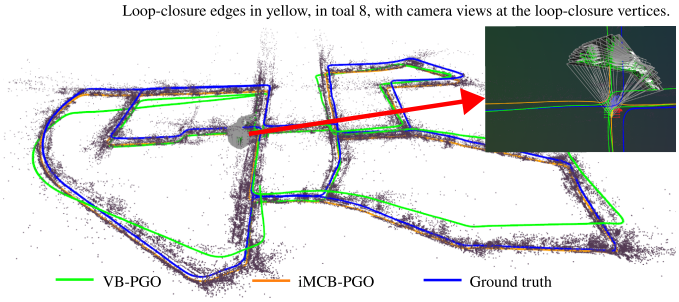


Fig. 8. iMCB-PGO on the KITTI dataset (stereo sequence 00) with one iteration allowed. The final costs of iMCB-PGO and VB-PGO are 2.17 and 2.67, respectively.

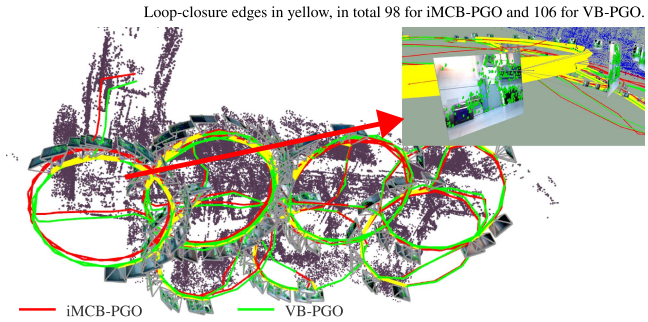


Fig. 9. iMCB-PGO on the Orazio dataset (RGB-D) with one iteration allowed. The final costs of iMCB-PGO and VB-PGO are 2915 and 3352, respectively.

than Cholesky factorization (i.e., solving linear systems in optimization).

Being faster to converge and faster to compute than VB-PGO, the iMCB-PGO pushes the research of CB-PGO [4] further in the case of real-time SLAM applications.

D. Application to SLAM System

We integrate iMCB-based PGO solver into proSLAM, a stereo/RGB-D visual SLAM system [15].

It is expected with sufficient convergence, both iMCB-PGO and VB-PGO yield similar results. However, we observe that iMCB-PGO converges faster. An example[2] is shown in Fig. 8, where only one iteration is allowed for all optimizers. Our iMCB-PGO achieves significantly better accuracy owing to its faster convergence. Lastly, we evaluate iMCB-PGO on the RGB-D dataset Orazio shown in Fig. 9. Since Orazio does not provide ground-truth poses, we report the cost after PGO optimization. It is worth noting that in real SLAM systems, the pose estimates can in turn impact tracking and loop-closure routines, resulting in different pose graphs. In fact, by setting maximum iterations to 1 and 3, VB-PGO would lead to different pose graphs for the lack of sufficient convergence with one iteration, while iMCB-PGO gives the same pose graphs owing to its faster convergence.

[2]Figs. 8 and 9 are generated by rerun.io.

VI. CONCLUSION

We proposed iMCB, an exact algorithm for incremental MCB computation. Our core contribution comprises a set of update rules for consistent shortest paths, which significantly reduces the complexity compared to batch-MCB [4]. We give proofs for the updates of both the consistent shortest paths and the isometric set, grounding our approach as an exact method. We show how iMCB can benefit real-time SLAM applications by proposing iMCB-PGO and demonstrating its potential using the proSLAM system. While we show results using PGO, we expect many other graph applications can benefit from our results in the future.

APPENDIX

Proof of Theorem 1: The loop-closure edge e_{uw} induces two potential shorter paths between arbitrary vertices s and d, in order of $v_s - v_u - v_w - v_d$ and $v_s - v_w - v_u - v_d$, respectively. Let $\omega_1 = \omega_{su} + \omega'_{uw} + \omega_{wd}$ and $\omega_2 = \omega_{sw} + \omega'_{uw} + \omega_{ud}$ be the total weights of these two paths. The LCR ensures consistency of P'_{sd} under $\omega_1 = \omega_2$ [4]. Then we prove that P'_{sd} is

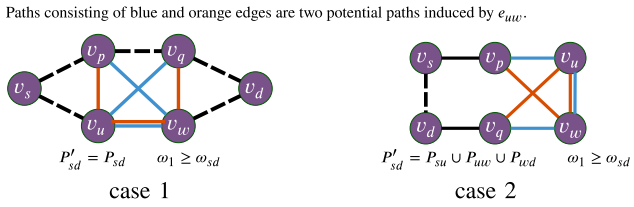


Fig. 10. We consider the impact of edge e_{uw} on the shortest path between v_s and v_d. The new shortest path may pass through e_{uw}, i.e., case 2, and may not, i.e., case 1.

consistent when $\omega_1 \neq \omega_2$ following the OWR. Without loss of generality, we assume $\omega_1 < \omega_2$. For P'_{sd} in G$'$, we consider two cases: $\omega_1 \geq \omega_{sd}$ and $\omega_1 < \omega_{sd}$.

Case 1: $\omega_1 \geq \omega_{sd}$, as shown in Fig. 10(a). By the OWR, we have $P'_{sd} = P_{sd}$. For any pair of vertices v_p, v_q (in order of $v_s - v_p - v_q - v_d$) on P'_{sd}, we have $P_{pq} \subseteq P_{sd}$ in G. Then, we prove $P'_{pq} = P_{pq}$ by justifying (9) and (10).

$$\omega_{pu} + \omega'_{uw} + \omega_{wq} \geq \omega_{pq} \tag{9}$$

$$\omega_{pw} + \omega'_{uw} + \omega_{uq} \geq \omega_{pq} \tag{10}$$

Assume (9) doesn't hold. Along path $\tilde{P}_{sd} = P_{sp} \cup P_{pu} \cup e_{uw} \cup P_{wq} \cup P_{qd}$, we observe

$$\begin{aligned} \omega_1 &= \omega_{su} + \omega'_{uw} + \omega_{wd} \\ &\leq \omega_{sp} + \omega_{pu} + \omega'_{uw} + \omega_{wq} + \omega_{qd} \\ &< \omega_{sp} + \omega_{pq} + \omega_{qd} = \omega_{sd} \end{aligned} \tag{11}$$

which contradicts $\omega_1 \geq \omega_{sd}$. We can prove (10) in a similar way. From (9) and (10), the new $p - q$ paths in G$'$ are always longer or equal than the old $p - q$ shortest path in G. Thus $P'_{pq} = P_{pq}$ by the OWR.

Case 2: $\omega_1 < \omega_{sd}$, as shown in Fig. 10(b). In this case, a strictly shorter $v_s - v_d$ path is found, and updated in G$'$ as $P'_{sd} = P_{su} \cup e_{uw} \cup P_{wd}$. We prove that for any pair of vertices v_p, v_q (in order of $v_s - v_p - v_q - v_d$) with v_p on P_{su} and v_q on P_{wd} (which is the only nontrivial configuration), the following inequality holds:

$$\omega_{pu} + \omega'_{uw} + \omega_{wq} < \omega_{pq} \tag{12}$$

We only need to inspect the path $P_{pq} = P_{pu} \cup e_{uw} \cup P_{wq}$ instead of $P_{pq} = P_{pw} \cup e_{uw} \cup P_{uq}$ due to the fact

$$\omega_{pw} + \omega'_{uw} + \omega_{uq} > \omega_{pu} + \omega'_{uw} + \omega_{wq}, \tag{13}$$

which can be easily proved by contradiction.

Assume (12) doesn't hold, i.e., $\omega_{pu} + \omega'_{uw} + \omega_{wq} \geq \omega_{pq}$. Along the path $P_{sp} \cup P_{pq} \cup P_{qd}$, we observe

$$\begin{aligned} \omega_{sd} &\leq \omega_{sp} + \omega_{pq} + \omega_{qd} \\ &\leq \omega_{sp} + \omega_{pu} + \omega'_{uw} + \omega_{wq} + \omega_{qd} \\ &= \omega_{su} + \omega'_{uw} + \omega_{wd} = \omega_1 \end{aligned} \tag{14}$$

which contradicts $\omega_1 < \omega_{sd}$.

According to (12), $P'_{pq} = P_{pu} \cup e_{uw} \cup P_{wq}$ is the shortest path from v_p to v_q in G$'$. As mentioned before, given $P'_{sd} = P_{su} \cup e_{uw} \cup P_{wd}$ and the fact that vertex v_p is on the path P_{su} and vertex v_q is on the path P_{wd}, we have

$$P'_{sd} = P_{sp} \cup P_{pu} \cup e_{uw} \cup P_{wq} \cup P_{qd} = P_{sp} \cup P'_{pq} \cup P_{qd}.$$

Thus $P'_{pq} \subseteq P'_{sd}$ holds for any p and q, confirming the consistency of P'_{sd} in G$'$.

REFERENCES

[1] C. Cadena et al., "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[2] F. Bai, S. Huang, T. Vidal-Calleja, and Q. Zhang, "Incremental SQP method for constrained optimization formulation in SLAM," in *2016 14th Int. Conf. Control, Automat. Robot. Vis.*, 2016, pp. 1–6.

[3] F. Bai, T. Vidal-Calleja, and S. Huang, "Robust incremental SLAM under constrained optimization formulation," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1207–1214, Apr. 2018.

[4] F. Bai, T. Vidal-Calleja, and G. Grisetti, "Sparse pose graph optimizati in cycle space," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1381–1400, Oct. 2021.

[5] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 2262–2269.

[6] G. Grisetti et al., "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," *Robot. Sci. Syst.*, vol. 3, 2007, pp. 65–72.

[7] S. Huang, Y. Lai, U. Frese, and G. Dissanayake, "How far is slam from a linear least squares problem?," in *2010 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 3011–3016.

[8] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G ^{2} o: A general framework for graph optimization," in *2011 IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3607–3613.

[9] F. Dellaert et al., "Factor graphs for robot perception," *Foundations Trends Robot.*, vol. 6, no. 1-2, pp. 1–139, 2017.

[10] V. Ila, L. Polok, M. Solony, and P. Svoboda, "SLAM++-A highly efficient and temporally scalable incremental SLAM framework," *Int. J. Robot. Res.*, vol. 36, no. 2, pp. 210–230, 2017.

[11] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres solver," Oct. 2023. [Online]. Available: https://github.com/ceres-solver/ceres-solver

[12] C. Estrada, J. Neira, and J. D. Tardós, "Finding good cycle constraints for large scale multi-robot slam," in *2009 IEEE Int. Conf. Robot. Automat.*, 2009, pp. 395–402.

[13] F. Bai, "Two novel techniques for graph optimization–cycle based formulation and change of optimal values," Ph.D. dissertation, Fac. Eng. Inf. Technol., 2020.

[14] B. Forsgren, K. Brink, P. Ganesh, and T. W. McLain, "Incremental cycle bases for cycle-based pose graph optimization," *IEEE Robot. Automat. Lett.*, vol. 8, no. 2, pp. 1021–1028, Feb. 2023.

[15] D. Schlegel, M. Colosi, and G. Grisetti, "ProSLAM: Graph SLAM from a programmer's perspective," in *2018 IEEE Int. Conf. Robot. Automat.*, 2018, pp. 3833–3840.

[16] T. Kavitha et al., "Cycle bases in graphs characterization, algorithms, complexity, and applications," *Comput. Sci. Rev.*, vol. 3, no. 4, pp. 199–243, 2009.

[17] J. D. Horton, "A polynomial-time algorithm to find the shortest cycle basis of a graph," *SIAM J. Comput.*, vol. 16, no. 2, pp. 358–366, 1987.

[18] J. dePina, "Applications of shortest path methods," Ph.D. dissertation, Univ. Amsterdam, 1995.

[19] E. Amaldi, C. Iuliano, and R. Rizzi, "Efficient deterministic algorithms for finding a minimum cycle basis in undirected graphs," in *Proc. Integer Program. Combinatorial Optim. 14th Int. Conf.*, Lausanne, Switzerland, Jun. 9-11, 2010, vol. 14, pp. 397–410.

[20] K. Mehlhorn and D. Michail, "Implementing minimum cycle basis algorithms," *J. Exp. Algorithmics*, vol. 11, pp. 2–5, 2007.

[21] E. Amaldi, C. Iuliano, T. Jurkiewicz, K. Mehlhorn, and R. Rizzi, "Breaking the O $(m^2 n)$ barrier for minimum cycle bases," in *Proc. Algorithms-ESA 17th Annu. Eur. Symp.*, Copenhagen, Denmark, Sep. 7–9, 2009, vol. 17, pp. 301–312.

[22] D. Hartvigsen and R. Mardon, "The all-pairs min cut problem and the minimum cycle basis problem on planar graphs," *SIAM J. Discrete Math.*, vol. 7, no. 3, pp. 403–418, 1994.