Sparse Pose Graph Optimization in Cycle Space

Fang Bai, *Member, IEEE*, Teresa Vidal-Calleja, *Member, IEEE*, and Giorgio Grisetti, *Member, IEEE*

***Abstract*—The state-of-the-art modern pose-graph optimization (PGO) systems are vertex based. In this context, the number of variables might be high, albeit the number of cycles in the graph (loop closures) is relatively low. For sparse problems particularly, the cycle space has a significantly smaller dimension than the number of vertices. By exploiting this observation, in this article, we propose an alternative solution to PGO that directly exploits the cycle space. We characterize the topology of the graph as a cycle matrix, and reparameterize the problem using relative poses, which are further constrained by a cycle basis of the graph. We show that by using a minimum cycle basis, the cycle-based approach has superior convergence properties against its vertex-based counterpart, in terms of convergence speed and convergence to the global minimum. For sparse graphs, our cycle-based approach is also more time efficient than the vertex-based. As an additional contribution of this work, we present an effective algorithm to compute the minimum cycle basis. Albeit known in computer science, we believe that this algorithm is not familiar to the robotics community. All the claims are validated by experiments on both standard benchmarks and simulated datasets. To foster the reproduction of the results, we provide a complete open-source C++ implementation[1] of our approach.**

***Index Terms*—Minimum cycle basis, pose graph optimization (PGO), special Euclidean group (SE(3)), simultaneous localization and mapping (SLAM).**

I. Introduction

POSE graph optimization (PGO) is a fundamental problem, which arises in various research disciplines, such as simultaneous localization and mapping (SLAM) [1]–[5], structure from motion [6]–[8], calibration of multicamera rig [9], and sensor network localization [10], [11].

A pose graph is a graph whose vertices encode positions and orientations of 3-D poses, and whose edges represent spatial constraints between the connected vertices. Taking a graph-based SLAM system as an example, the system processes the raw measurements to construct local maps. These local maps are then arranged in a pose graph as vertices. Constraints between local maps arise from matching nearby local maps or from proprioceptive measurements coming from odometry or inertial measurement units.

Typically the constraints are affected by some uncertainty, which is modeled as a Gaussian (or Langevin) distribution centered around the equilibrium point of the constraint. For example, in graph-based SLAM, systematic biases, noise in the sensors, errors in localization propagate to the estimation of these constraints. Hence, in real applications, it is impossible to find a configuration of vertices that simultaneously nulls the residual error of all constraints. PGO is then the task finding the configuration of the vertices that is maximally consistent with the constraints (i.e., edges), via solving a nonlinear least squares optimization problem.

In real SLAM applications, the number of edges is typically proportional to the number of vertices. This is a consequence of the local nature of SLAM, stemming from the limits in the sensor range. Only local maps that are spatially close can share some common elements, and thus, the corresponding vertices can be connected by constraints (i.e., edges). This results in limiting the number of edges connected to a vertex, and ultimately leads to a sparsely connected graph. By leveraging on this sparsity, modern PGO systems [12]–[15] are capable of solving extremely large problems in a fraction of seconds. We term the method in [12]–[15] as vertex-based approaches for a reason that will explain later on.

At its core, the state-of-the-art PGO techniques solve a sparse linear system to update the estimates of vertices [12]–[15]. This system is typically solved by a sparse Cholesky factorization [16], which is guided by the graph topology presented as a vertex-edge incidence matrix. In graph theory, the incidence matrix spans a space called cut space, which is orthogonal complementary to a space called cycle space. A sparse graph implies the following two facts: (a) low connectivity between vertices, which has been reflected in the incidence matrix and exploited effectively [16]; (b) low dimensionality of cycle space, which has been largely ignored due to the huge success of sparse Cholesky factorization with respect to vertices. In this article, we will show the possibility of designing effective PGO techniques in the cycle space.

To this end, we reformulate the conventional least squares optimization with a relative parametrization, i.e., using relative poses (associated with edges), as variables to be estimated. This induces the overparameterization of the problem since the number of edges is higher than that of vertices. This issue is solved by introducing a collection of inherent constraints that the value of vertices anchors all the paths between any

Manuscript received April 29, 2020; accepted December 8, 2020. This work was supported in part by the University of Technology, Sydney, and in part by the China Scholarship Council. This article was recommended for publication by Associate Editor J. M. M. Montiel and Editor F. Chaumette upon evaluation of the reviewers' comments. *(Corresponding author: Fang Bai.)*

Fang Bai and Teresa Vidal-Calleja are with the Centre for Autonomous Systems (CAS), University of Technology Sydney, Ultimo, NSW 2007, Australia (e-mail: fang.bai@yahoo.com; teresa.vidalcalleja@uts.edu.au).

Giorgio Grisetti is with the Department of Computer, Control, and Management Engineering "Antonio Ruberti," Sapienza University of Rome, Rome 00185, Italy (e-mail: grisetti@dis.uniroma1.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRO.2021.3050328.

Digital Object Identifier 10.1109/TRO.2021.3050328

[1]Code: https://bitbucket.org/FangBai/cyclebasedpgo

two vertices to generate the same composed transformation. Such a set of constraints are topologically characterized by the graph cycle space, which can be described by a cycle basis. Finally, our optimization problem is casted as a constrained least-squares optimization problem, which can be solved in the sequential quadratic programming (SQP) scheme. We term this PGO technique as cycle-based approach.

A common issue in the relative parameterization is that it is not necessarily sparse [17]–[20]. It turns out this issue can be resolved by using a minimum cycle basis (MCB); however, the computation of an MCB itself is a hard problem. We exploit the fact that graphs in PGO (and other similar applications) are sparse, with positive (integer) weights, to design a tailored MCB algorithm that can greatly mitigate this issue, in particular for sparse graphs that are encountered in real SLAM/PGO applications. It can be shown that relative formulations have faster convergence compared with the vertex-based ones in the absolute frame (both in this work and the work in [20]). Therefore, for sparse graphs, based on the MCB, the cycle-based approach can attain faster (or comparable at least) computational time compared with the vertex-based ones, due to the reduced dimension in the cycle space and the sparsity forced by the MCB. Aside from the numerical sparsity, the usage of the MCB can also improve the convergence to the global minimum.

Concretely, we make following contributions in this article.

1) We propose a cycle-based PGO (CB-PGO) formulation based on the cycle matrix and $\mathrm{SE}(3)$ Lie group, and derive an SQP algorithm on the manifold to solve it.
2) We give insights that the matrix structure in the Cholesky factorization is characterized by a cycle matrix: (a) the matrix to be factorized has exactly the same dimension as that in the cycle space; (b) the numerical sparsity can be maximized by an MCB.
3) We propose an effective MCB algorithm that is tailored for sparse graphs with positive integer weights. We give theoretical insights such as LexDijkstra, and working heuristics such as pruning vertices of degree two.
4) We provide principled analyses in terms of observability, convergence with respect to cycle bases, and convergence rate for the CB-PGO formulation.
5) We provide extensive experimental results to validate the advantages of using CB-PGO, in terms of both the computational complexity and robustness.
6) We provide a C++ implementation of the overall algorithm, which is freely available to the community.

The remainder of this article is structured as follows. Section II reviews the related work. The Lie group and graph preliminaries are provided in Section III. Section IV recaps the conventional vertex-based PGO (VB-PGO) formulation. Section V derives the CB-PGO formulation and the corresponding SQP solver on the manifold. Section VI is dedicated to calculate an MCB for sparse graphs with positive integer weights. Analyses on the observability and convergence are presented in Section VII. Details of our C++ implementation are provided in Section VIII. Experimental validations are given in Section IX. Section X concludes this article.

II. Related Work

PGO, as a maximum likelihood estimation (MLE), was first described in the seminal paper by Lu *et al.* [21], where a nonlinear least squares (NLS) is used to optimize the network generated by scan-matchings. At the time, although in theory, techniques like Gauss–Newton [22] were available to solve the NLS problem, the development of its numerical side was a bit behind. To address the computational complexity, Frese *et al.* [23] proposed a multilevel relaxation method, based on the Gauss–Seidel relaxation. Olson *et al.* [24] suggested an incremental pose parameterization, and a PGO solver based on the stochastic gradient descent method, which had a large basin of convergence to the global optimum. Grisetti *et al.* [25] extended the framework to 3-D, and applied a tree parameterization to improve the convergence speed.

The rapid advancement in sparse linear algebra techniques (see Davis [16]) completely changed the landscape. In robotics, Dellaert *et al.* [12] are the first to show that MLE can be solved efficiently by a Gauss–Newton method, using sparse matrix decompositions. Kaess *et al.* [26] attributed to the incremental solver of MLE, using the Givens rotation based QR decomposition, or the Bayes tree [27]. Kummerle *et al.* [13] designed a general graph based optimization framework. Ila *et al.* [14] exploited the block structure of sparse matrices. Besides matrix decompositions, the resulting linear system can also be solved by an iterative method, for example, preconditioned conjugate gradient [28]–[30]. The convergence property of the Gauss–Newton based method can be further improved by using the idea of trust region [22], like Levenberg–Marquardt [31], or Powell's-dog-leg [32]. All these MLE techniques can provide rather efficient PGO solutions.

It is possible to exploit specific structures of PGO to design more specialized solvers, for example, the divide-and-conquer methods by Grisetti *et al.* [33], [34]. The basic idea is to divide the full PGO into several submaps (i.e., subgraphs), solve each one of them, and then join the submaps together to obtain an approximation to the full PGO. Zhao *et al.* [35], [36] investigated the special case of joining two submaps, with a clever parameterization, which can be solved by a linear least squares, followed by a nonlinear transformation. For 2-D cases, Carlone *et al.* [37] suggested a linear approximation framework to PGO, by computing first an orientation estimation, and then the position part using the given orientation. The core was the regularization of rotation angles [37], which was systematically addressed in [38] using a quadratic integer programming. The separability of the orientation and position estimation was further studied by Khosoussi *et al.* [39], based on a variable projection approach.

Besides practical algorithms to solve PGO, some theoretical insights are also drawn. Huang *et al.* [17] showed empirically that a point-feature-based SLAM is close to a convex optimization problem, and a relative formulation is proposed for the purpose of reducing the nonlinearity in SLAM. Wang *et al.* [40] discussed the number of local minimums for PGO in special cases. Carlone [41] provided an analysis on the convergence basin of the global minimum for the Gauss–Newton method. Several key factors are concluded, for example, orientation noises, and graph topologies. With the assumption of isometric additive Gaussian noise, Khosoussi *et al.* [42] established the connection between the Fisher information matrix (FIM) of the estimate and the graph complexity.

Convex optimization is a powerful technique to design globally optimal solutions to PGO. An early touch on this topic came from Liu *et al.* [43], who relaxed planar PGO into a semidefinite programming (SDP), which was solved by standard convex optimization tools [44]. It can be observed that the nonconvexity of PGO comes from the cost function, and manifold constraints. Carlone *et al.* [15] showed that the cost function can be chosen convex by using a Frobenius norm with an isotropic noise model, [45]–[48]. The manifold constraints can be relaxed by their convex-hulls, as shown by Rosen *et al.* [48]. However, the relaxations in [43] and [48] are not tight enough. Carlone *et al.* [45], [47] explored the Lagrangian duality of PGO, leading to a tight SDP relaxation, which can be verified to be globally optimal in many cases. Rosen *et al.* [15], designed a certifiable PGO solver, exploiting convex relaxations, and a Riemannian trust-region method. Briales *et al.* [50] suggested a compact matrix formulation, with concise and efficient derivations.

Loop-closing constraints and cycles have a rich history in SLAM literatures. Estrada *et al.* [51] formulated the loop-closing problem between local maps as a quadratic optimization problem with equality constraints. The problem was solved by SQP, and a connection to iterated extended Kalman filter was drawn. Russell *et al.* [52] proposed a distributed network optimization method based on the graph cycle space, and proved its convergence in linear cases. Olson [53] evaluated the pairwise consistency of two loop-closing edges by joining them with the odometry sequence as a cycle. Dubbelman *et al.* [54] employed interpolations in $\mathrm{SE}(3)$ along a cycle to obtain an approximate solution to pose-chain SLAM. The concept of cycle bases was used by Carlone *et al.* in [37], [38], [41]. The loop-closing cycle in a point-feature-based SLAM was considered by Bai *et al.* [18], while both point and line-features are included by Guo *et al.* [55] in more specific scenarios. Later, Bai *et al.* [19] formulated PGO explicitly as a constrained optimization problem by using cycles in the graph. The cycle structure in graph optimization is typically presented as relative formulations [18], [19], which have been used in the work [17], [20], [24], [25], [56]–[60] as well.

Although the usage of concepts such as cycle space and cycle bases abounds in existing literatures, none of these works study how to design efficient cycle-based optimization algorithms by exploiting: the dimension reduction of Cholesky factorization in the cycle space due to graph sparsity, and the possibility of designing a tailored MCB algorithm that takes advantage of sparsity and positive integer weights.

III. Preliminaries and Backgrounds

A. Notations

For any two sets $\mathcal{X}_1$ and $\mathcal{X}_2$, we denote, respectively, $\mathcal{X}_1 \cap \mathcal{X}_2$ the intersection, $\mathcal{X}_1 \cup \mathcal{X}_2$ the union, $\mathcal{X}_1 \backslash \mathcal{X}_2$ the difference, and $\mathcal{X}_1 \oplus \mathcal{X}_2 = (\mathcal{X}_1 \cup \mathcal{X}_2) \backslash (\mathcal{X}_1 \cap \mathcal{X}_2)$ the symmetric difference of these two sets. Let $|\mathcal{X}|$ be the cardinality of the set $\mathcal{X}$. We will use $\mathbb{Z}$ to denote the set of integers, and $\mathbb{R}$ denotes the set of real numbers. If not explicitly stated, the lower-case in normal font, the lower-case in bold font, and the upper-case in bold font are reserved for scalars, vectors, and matrices, respectively. A matrix of zeros is denoted by $\mathbf{O}$, and an identity matrix is denoted by $\mathbf{I}$. $\mathbf{A}^T$ represents the transpose, and $\mathbf{A}^\dagger$ represents the Moore–Penrose pseudo inverse of a matrix $\mathbf{A}$. The notation $\{\mathbf{v}_i\}^\perp$ stands for the orthogonal complement to the space spanned by a set of vectors $\{\mathbf{v}_i\}$. $<\mathbf{v}_1, \mathbf{v}_2> = \mathbf{v}_1^T \mathbf{v}_2$ is the inner product between $\mathbf{v}_1$ and $\mathbf{v}_2$. The squared Mahalanobis distance is denoted by $\|\mathbf{e}\|_{\mathbf{\Sigma}}^2 = \mathbf{e}^T \mathbf{\Sigma}^{-1} \mathbf{e}$. The notation $[m:n]$ is used to describe a sequence of consecutive integers from m to n. We will use "iff" as a shorthand of "if and only if." The graph notations used throughout this article are listed in Table I.

TABLE I
List of Notations on Graph

$\mathbb{GF} = \mathbb{Z}_2$		finite field of modulo 2
$\mathbb{GF}^{d_1 \times d_2}$		$d_1 \times d_2$ dimensional matrix on $\mathbb{Z}_2$
$\mathcal{G} = \mathcal{G}(\mathcal{V}, \mathcal{E})$		undirected graph
$\mathcal{E}$		edge set of a graph $\mathcal{G}$
$\mathcal{V}$		vertex set of a graph $\mathcal{G}$
e_{uv}	$\mathcal{E}$	edge with endpoints u, v
$\nu = \|\mathcal{E}\| - \|\mathcal{V}\| + 1$	$\mathbb{R}$	dimension of cycle space
$\mathcal{T} = \mathcal{T}(\mathcal{G})$	$\mathbb{GF}^{1 \times m}$	tree in $\mathcal{G}$
$\mathcal{P}_{uv} = \mathcal{P}_{uv}(\mathcal{G})$	$\mathbb{GF}^{1 \times m}$	path in $\mathcal{G}$, from vertex u to v
$\mathcal{C} = \mathcal{C}(\mathcal{G})$	$\mathbb{GF}^{1 \times m}$	cycle in $\mathcal{G}$
$\mathcal{S}$	$\mathbb{GF}^{1 \times m}$	support vector
Note: A vector on $\mathbb{GF}^{1 \times m}$ can be sparsely described by a set.		
$\{*\}$		set with elements in form of $*$
$\boldsymbol{\mathcal{H}}$	$\{\mathcal{C}\}$	Horton set
$\boldsymbol{\mathcal{I}}$	$\{\mathcal{C}\}$	set of isometric cycles
$\boldsymbol{\mathcal{B}}$	$\{\mathcal{C}\}$	cycle basis
$\mathcal{B} = \mathcal{B}(\mathcal{G})$	$\mathbb{GF}^{n \times \nu}$	cycle matrix of cycle basis $\boldsymbol{\mathcal{B}}$
$\mathbf{T}_k$	$\mathrm{SE}(3)$	poses
$\mathbf{T}_{\mathbf{k}}$	$\mathrm{SE}(3)$	relative poses
$\boldsymbol{\mathcal{P}}_{uv}$	$\{\mathbf{T}_{\mathbf{k}}\}$	geometric path from u to v
$\boldsymbol{\mathcal{C}}$	$\{\mathbf{T}_{\mathbf{k}}\}$	geometric cycle

B. Special Euclidean Group

The special Euclidean group, $\mathrm{SE}(3)$, is a standard tool to describe rigid-body transformations [61], [62], which typically occur in robotics and computer vision community.

$\mathrm{SE}(3)$ is a Lie group. A Lie group is a peculiar smooth manifold whose local structure can be described by the so-called Lie algebra, which is the tangent space at the identity of the group. Let $\mathfrak{se}(3)$ be the Lie algebra of $\mathrm{SE}(3)$. Both $\mathrm{SE}(3)$ and $\mathfrak{se}(3)$ can be described by matrices. For each matrix $\mathbf{T} \in \mathrm{SE}(3)$, we can find an associated matrix $\mathbf{X} \in \mathfrak{se}(3)$, and vice versa, by matrix exponential and matrix logarithm: $\mathbf{T} = \mathbf{exp}(\mathbf{X})$, $\mathbf{X} = \mathbf{log}(\mathbf{T})$.

An element $\mathbf{X} \in \mathfrak{se}(3)$ can be identified by a "screw matrix," taking the form

$$\mathbf{X} = \begin{bmatrix} 0 & -x_3 & x_2 & x_4 \\ x_3 & 0 & -x_1 & x_5 \\ -x_2 & x_1 & 0 & x_6 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

It is obvious that each screw matrix $\mathbf{X}$ can be uniquely identified by a vector $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]^T \in \mathbb{R}^6$. The relationship can be expressed as $\mathbf{X} = \mathbf{x}^\wedge$, $\mathbf{x} = \mathbf{X}^\vee$ with operation $\wedge$ and $\vee$. Therefore, for convenience, we define an encapsulated exponential and logarithm mapping between $\mathbf{T}$ and $\mathbf{x}$ directly as

$$\mathbf{T} = \mathbf{Exp}(\mathbf{x}) = \mathbf{exp}(\mathbf{x}^\wedge), \ \mathbf{x} = \mathbf{Log}(\mathbf{T}) = \mathbf{log}(\mathbf{T}^\vee).$$

The adjoint of Lie algebra, $\mathbf{ad}(\mathbf{x})$, is related to a binary operation $[\cdot, \cdot]$, called Lie bracket, yielding the relation

$$[\mathbf{X}, \mathbf{Y}] = \mathbf{XY} - \mathbf{YX} = (\mathbf{ad}(\mathbf{x})\mathbf{y})^\wedge$$

which holds for any $\mathbf{Y} \in \mathfrak{se}(3)$, $\mathbf{y} = \mathbf{Y}^\vee \in \mathbb{R}^6$. Exponentiating $\mathbf{ad}(\mathbf{x})$, we would get a matrix $\mathbf{Ad}(\mathbf{T})$ called the adjoint of Lie group. The adjoint matrix has a nice property

$$\mathbf{T} \cdot \mathbf{Exp}(\mathbf{y}) = \mathbf{Exp}\left(\mathbf{Ad}\left(\mathbf{T}\right)\mathbf{y}\right)\mathbf{T} \tag{1}$$

which can be used to shift the position of $\mathbf{T}$ and $\mathbf{Exp}(\cdot)$. Another property of $\mathbf{Ad}(\cdot)$ is

$$\mathbf{Ad}(\mathbf{T}_1)\mathbf{Ad}(\mathbf{T}_1) = \mathbf{Ad}(\mathbf{T}_1\mathbf{T}_2)$$

which is used to collect two $\mathbf{Ad}(\cdot)$ together.

The Baker–Campbel–Hausdorff formula (BCH) is used to concatenate two matrix exponentials. The exact BCH formula is expressed as a series, and a closed form approximation is

$$\mathbf{Exp}\left(\mathbf{x}\right)\mathbf{Exp}\left(\mathbf{y}\right) \approx \begin{cases} \mathbf{Exp}\left(\mathbf{J}_\mathbf{l}^{-1}\left(\mathbf{y}\right)\mathbf{x} + \mathbf{y}\right), & \text{if } \mathbf{x} \to \mathbf{0} \\ \mathbf{Exp}\left(\mathbf{x} + \mathbf{J}_\mathbf{r}^{-1}\left(\mathbf{x}\right)\mathbf{y}\right), & \text{if } \mathbf{y} \to \mathbf{0} \end{cases}$$

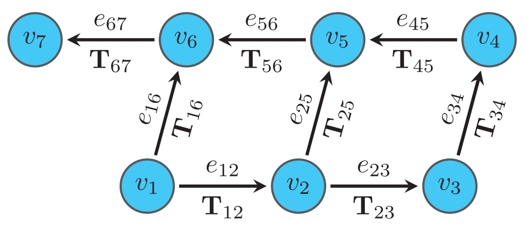
where $\mathbf{J}_\mathbf{l}(\cdot)$ and $\mathbf{J}_\mathbf{r}(\cdot)$ are called the left-hand and right-hand Jacobian of the exponential coordinate parameterization.

The mappings between $\mathrm{SE}(3)$ and $\mathfrak{se}(3)$, the adjoint operation and the BCH formula are used to linearize PGO, which is the prerequisite to apply an iterative nonlinear solver.

For $\mathrm{SE}(3)$, all the operations $\mathbf{Exp}(\cdot)$, $\mathbf{Log}(\cdot)$, $\mathbf{ad}(\cdot)$, $\mathbf{Ad}(\cdot)$, $\mathbf{J}_\mathbf{l}(\cdot)$, and $\mathbf{J}_\mathbf{r}(\cdot)$ are calculated in closed form [61], [62].

C. Graph Theory

Let $\mathcal{G}$ be an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set, and $\mathcal{E}$ is a set of unordered pairs (u, v), with $u, v \in \mathcal{V}$. The elements in $\mathcal{V}$ are termed vertices (or nodes), and the elements in $\mathcal{E}$ are termed edges. In what follows, we will denote an edge from u to v as e_{uv}. An edge e_{uv} is said to be incident to the vertices u and v, while u and v are called the endpoints of e_{uv}. The degree of a vertex in $\mathcal{G}$ is the number of edges incident to that vertex. A subgraph of $\mathcal{G}$ stands for a graph with only part of vertices and edges from $\mathcal{G}$. In particular, we will be interested in three types of subgraphs, i.e., path, cycle, and tree. Formally, a graph is said to be connected if there exists a path for any pair of vertices in the graph. A path is a connected subgraph in which there are exactly two vertices having degree of one, and the rest of vertices having degree of two. A cycle is a subgraph in which every vertex has an even degree. If a cycle is connected and the degree of each vertex is exactly two, the cycle is called a simple/elementary cycle, or a circuit. A tree is a connected subgraph, which contains no cycles (i.e., acyclic subgraph). If a tree of $\mathcal{G}$ contains all vertices in $\mathcal{V}$, it is called a spanning tree of $\mathcal{G}$. We will use $\mathcal{P}$ to denote a path, $\mathcal{C}$ to denote a cycle, and $\mathcal{T}$ to denote a tree, respectively.



$$\mathcal{B} = \begin{array}{c|cccccccc} & e_{12} & e_{23} & e_{34} & e_{45} & e_{56} & e_{67} & e_{25} & e_{16} \\ \hline \mathcal{C}_1 & 1 & & & & 1 & & 1 & 1 \\ \mathcal{C}_2 & & 1 & 1 & 1 & & & 1 & \end{array}$$

Fig. 1. Toy graph of PGO. For each relative poses $\mathbf{T}_{i,j}$, the edge e_{ij} is oriented as $i \rightarrowtail j$. When talking about topological information, such as cycles/paths, we can safely operate on the undirected version by ignoring the edge orientations, and lifting back to oriented edges when the cycles/paths are computed. In a graph, paths/cycles are a collection of edges, which can be described by a set or a vector on $\mathbb{GF}$. For example, there are three simple cycles in this graph. $\mathcal{C}_1 = \{e_{12}, e_{25}, e_{56}, e_{16}\}$, $\mathcal{C}_2 = \{e_{23}, e_{34}, e_{45}, e_{25}\}$, and $\mathcal{C}_3 = \{e_{12}, e_{23}, e_{34}, e_{45}, e_{56}, e_{16}\}$. Cycles can be concatenated by the symmetric difference of sets: $\mathcal{C}_3 = \mathcal{C}_1 \oplus \mathcal{C}_2$. In this graph, $\mathcal{C}_1$ and $\mathcal{C}_2$ are two independent cycles forming a cycle basis of the graph. The vectorized representation on $\mathbb{GF}$, i.e., the cycle basis matrix, is presented in $\mathcal{B}$, where the blanks are zeros. $\mathcal{C}_3$ can be written as a vector $\mathcal{C}_3 = [0, 1, 1, 1, 1, 1, 0, 0, 1]$. Based on the arithmetics of $\mathbb{GF}$, we have $\mathcal{C}_3 = \mathcal{C}_1 + \mathcal{C}_2$, which is in accordance with the symmetric difference of sets.

A subgraph, i.e., a path/cycle/tree, can be uniquely identified by the set of edges it used, which induces an "incidence vector" whose elements are assigned to either 0 or 1. For instance, a cycle $\mathcal{C}$ can be expressed as an incidence vector $[c_1, c_2, \ldots, c_{|\mathcal{E}|}]$, with $c_k = 1$ $(k = 1, 2, \ldots, |\mathcal{E}|)$ iff the kth edge is used by the cycle $\mathcal{C}$, and $c_k = 0$ otherwise (see Fig. 1). The concept of finite field (or Galois field) is useful to describe this phenomenon. A finite field is basically a finite set equipped with arithmetic rules. In particular, we are interested in the finite field of order 2, denoted as $\mathbb{GF} = \mathbb{Z}_2 = \{0, 1\}$, whose elements are 0 and 1 only. The addition and multiplication on $\mathbb{Z}_2$ are defined, respectively, to be the addition and multiplication on $\mathbb{Z}$ modulo 2. Obviously, incidence vectors (to describe paths/cycles/trees) are vectors on $\mathbb{GF}$. Moreover, all the cycles in $\mathcal{G}$ can be described by a matrix on $\mathbb{GF}$ with each row being a cycle incidence vector. This matrix is called cycle matrix: $\mathcal{B} = [b_{i,j}]$, with $b_{i,j} = 1$ iff the jth edge is contained in the ith cycle, and $b_{i,j} = 0$ otherwise. The rows of $\mathcal{B}$ span a vector space over the two-element finite field based on the modulo two arithmetics, which is called cycle space. A basis to the cycle space is called cycle basis. The cycle space of an undirected graph is orthogonal complementary to the so-called cut space. The cut space is not needed to understand this work, but important to build connections with Gauss–Newton based optimizers [12]–[14]. Interested readers are referred to [63], [64] for accessible explanations. For a connected graph, the cycle space has a dimension $\nu = |\mathcal{E}| - |\mathcal{V}| + 1$, and the cut space has a dimension $|\mathcal{V}| - 1$.

Let $\mathbf{x}_1$, $\mathbf{x}_2$ be two vectors on $\mathbb{GF}^{|\mathcal{E}|}$, and $\mathcal{X}_1$, $\mathcal{X}_2$ be the corresponding set representations. Then, the vector addition $\mathbf{x}_1 + \mathbf{x}_2$ on $\mathbb{GF}^{|\mathcal{E}|}$ corresponds to the symmetric difference of sets, i.e.,

$\mathcal{X}_1 \oplus \mathcal{X}_2$. The inner product satisfies: $< \mathbf{x}_1, \mathbf{x}_2 >= \mathbf{x}_1^T \mathbf{x}_2 = 1$ iff $|\mathcal{X}_1 \cap \mathcal{X}_2|$ is odd; $< \mathbf{x}_1, \mathbf{x}_2 >= \mathbf{x}_1^T \cdot \mathbf{x}_2 = 0$ iff $|\mathcal{X}_1 \cap \mathcal{X}_2|$ is even. In what follows, we will use the vector representation and set representation interchangeably and describe both with the same notation, where the difference can be easily told by the operation used.

IV. Traditional PGO

The topology of PGO can be visualized as a graph whose vertices represent poses. An edge is created if there exists a relative geometric relation between two poses, i.e., relative poses, which can be produced by a wheel-encoder, scan-matching [21], [65], [66], epipolar geometry [67], [68], or visual loop-closing techniques, etc., [69], [70].

Both poses and relative poses are rigid-body transformations, which can be described by $\mathrm{SE}(3)$ Lie group. Denote $\mathbf{T}_i$ to be the ith pose. The relative poses from the ith pose to the jth pose, denoted by $\mathbf{T}_{i,j}$, is a rigid-body transformation evaluated in the local coordinate frame of the ith pose, which mathematically writes $\mathbf{T}_{i,j} = \mathbf{T}_i^{-1}\mathbf{T}_j$. For the clarity of notations, we assign each edge a unique index, and use $\mathbf{T}_\mathbf{k}$ with subscript in bold to represent a relative poses.

For each relative pose $\mathbf{T}_\mathbf{k}$, there is a noisy measurement $\tilde{\mathbf{T}}_\mathbf{k}$. The measurement noise is conventionally assumed to be zero-mean Gaussian in the vector space of $\mathrm{SE}(3)$ Lie algebra, which can be mathematically formalized as

$$\mathbf{Log}(\tilde{\mathbf{T}}_\mathbf{k}^{-1} \cdot \mathbf{T}_\mathbf{k}) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_\mathbf{k}).$$

Note that other noise models are also possible, for example, the matrix Langevin distributions in [15] and [45].

Let the topology of PGO be described by the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Then, PGO aims to obtain a MLE for the set of poses $\{\mathbf{T}_i\}_{i\in\mathcal{V}}$ using the set of measurements of relative poses$\{\tilde{\mathbf{T}}_\mathbf{k}\}_{\mathbf{k}\in\mathcal{E}}$, via solving a least squares optimization problem

$$\{\mathbf{T}_i\}_{i\in\mathcal{V}} = \arg\min \sum_{\mathbf{k}\in\mathcal{E}} \|\mathbf{Log}(\tilde{\mathbf{T}}_\mathbf{k}^{-1}\mathbf{T}_\mathbf{k})\|^2_{\mathbf{\Sigma}_\mathbf{k}}. \tag{2}$$

Remark 1: Note that the relative poses $\mathbf{T}_{i,j}$ are evaluated at the local frame of the ith pose, so ideally a PGO is described by a directed graph. However, the edge orientation will not affect the topological side of a graph, such as paths/cycles we discuss later on. Therefore, we opt to describe PGO as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The restriction to the undirected graph limits the graph matrices/vectors to Galois field instead of real numbers.

Besides, the undirected edges can be easily lifted to directed edges whenever it is desired.

Remark 2: The traditional PGO is unobservable, in the sense that if $\{\mathbf{T}_i\}_{i\in\mathcal{V}}$ is a solution to (2), then $\forall \mathbf{T}' \in \mathrm{SE}(3)$, $\{\mathbf{T}'\mathbf{T}_i\}_{i\in\mathcal{V}}$ is also a solution to (2). This can be easily verified by the fact that $\mathbf{T}_{i,j} = \mathbf{T}_i^{-1}\mathbf{T}_j = (\mathbf{T}'\mathbf{T}_i)^{-1}(\mathbf{T}'\mathbf{T}_j)$, thus, $\{\mathbf{T}_i\}_{i\in\mathcal{V}}$ and $\{\mathbf{T}'\mathbf{T}_i\}_{i\in\mathcal{V}}$ yield exactly the same contribution in the cost function. To ensure a unique solution, a popular practice is to anchor some poses (usually the first pose) to a fixed value or the identity of $\mathrm{SE}(3)$.

V. Cycle-Based PGO

The state-of-the-art PGO techniques [12]–[15], [45] describe PGO as a factor graph [72], whose topology is represented by an incidence matrix. The graph is solved by a second-order optimization technique, for example, Gauss–Newton, which results to solve a sparse linear system whose dimension is decided by the number of vertices. These approaches solve PGO in the cut space, and we will term them as VB-PGO. The VB-PGO can be solved rather efficiently by sparse matrix factorizations [16].

Practical PGO instances are rather sparse. Let us measure the graph sparsity as a concept called cycle ratio, defined as $\frac{\nu}{|\mathcal{E}|}$, i.e., the dimension of the cycle space divided by the number of edges. Empirically, a PGO instance encountered in SLAM has a cycle ratio well below 20%, which implies $\frac{\nu}{|\mathcal{V}|-1} < 1/4$, i.e., the dimension of the cut space is at least four times larger than that of the cycle space. Let alone SLAM instances with a long trajectory and a few loop-closures, whose cycle ratio can be less than 5%, or even 1%.

In this section, we will describe an approach that transforms PGO from the cut space to the cycle space. The CB-PGO, denoted as CB-PGO, has a reduced dimension compared with its vertex-based counterpart, as long as the graph is sparse enough. While the dimension reduction to the cycle space can undermine the sparsity of PGO, we propose to maximize the sparsity by a minimum cycle basis which will be described in Section VI.

The observability and convergence properties of the proposed CB-PGO are discussed in Section VII.

A. Preliminaries

In what follows, we will use the term toplogical path and toplogical cycle to represent a path and cycle in a pure topological graph $\mathcal{G}$. If such a $\mathcal{G}$ is associated with geometric information, namely by associating vertices with poses and edges with relative poses, respectively, we will term this graph as a geometric graph. A (topological) path $\mathcal{P}$ whose edges are associated with relative poses will be termed as a geometric path, denoted by $\boldsymbol{\mathcal{P}}$. Analogously, a (topological) cycle $\mathcal{C}$ with edges associated with relative poses will be termed as a geometric cycle, denoted by $\boldsymbol{\mathcal{C}}$. If not explicitly stated, the terms, paths, and cycles, refer to the topological version.

The orientations of edges are irrelevant in this article when discussing the topology of $\mathcal{G}$, as well as concepts such as cycle bases and sparsity. However, they are useful in terms of describing the geometric paths/cycles. The orientation of an edge is decided by the geometric information, i.e., relative poses it associated with. For example, given an edge $\mathbf{k}$ in $\mathcal{G}$ with the associated relative poses being $\mathbf{T}_{i,j} = \mathbf{T}_i^{-1}\mathbf{T}_j$, we stipulate the edge orientation to be from i to j. In other words, $i \rightarrowtail j$ is the forward direction of the edge $\mathbf{k}$, and $j \rightarrowtail i$ is the backward direction.

B. Consistency of PGO

Let $\mathcal{P}_{st}$ be a path from s to t in $\mathcal{G}$. The corresponding geometric path $\boldsymbol{\mathcal{P}}_{st}$ is defined as

$$\boldsymbol{\mathcal{P}}_{st} = \mathbf{T}_{\mathbf{1}^p}^{\varrho(\mathbf{1}^p)}\mathbf{T}_{\mathbf{2}^p}^{\varrho(\mathbf{2}^p)}\cdots\mathbf{T}_{\boldsymbol{\theta}^p}^{\varrho(\boldsymbol{\theta}^p)} \tag{3}$$

where $\mathbf{T}_{\mathbf{1}^p}, \mathbf{T}_{\mathbf{2}^p}, \ldots, \mathbf{T}_{\boldsymbol{\theta}^p}$ is the sequence of relative poses by traversing $\mathcal{P}_{st}$ from s to t. In (3), the superscript $\varrho(\mathbf{k}^p)$ is assigned to $+1$ if the traversal uses the edge $\mathbf{k}^p$ in the forward direction, and -1 if in the backward direction. The superscript of a matrix $\mathbf{T}$ will be interpreted as the power of the matrix, by $\mathbf{T}^{+1} = \mathbf{T}$ and $\mathbf{T}^{-1} = \mathbf{inv}(\mathbf{T})$.

PGO is a consistent formulation with respect to poses and relative poses. To show this, let $\mathbf{T}_s$ and $\mathbf{T}_t$ be two arbitrary poses. Let $\boldsymbol{\mathcal{P}}_{1:st}$ and $\boldsymbol{\mathcal{P}}_{2:st}$ be two geometric paths from pose s to t. Then, the pose $\mathbf{T}_t$ calculated from these two paths are exactly the same, i.e., $\mathbf{T}_t = \mathbf{T}_s\boldsymbol{\mathcal{P}}_{1:st} = \mathbf{T}_s\boldsymbol{\mathcal{P}}_{2:st}$. Obviously, the consistency of the PGO can be also interpreted as the equivalence of the geometric paths, in the sense that $\boldsymbol{\mathcal{P}}_{1:st} = \boldsymbol{\mathcal{P}}_{2:st} = \mathbf{T}_s^{-1}\mathbf{T}_t$. At last, with a slightly abuse of notation, we can write the consistency of two geometric paths as $\boldsymbol{\mathcal{P}}_{1:st}^{-1}\boldsymbol{\mathcal{P}}_{2:st} = \mathbf{I}$, which is a geometric cycle.

The geometric cycles will play a key role in formulating PGO in cycle space, which in general ensures the consistency of the PGO. On the other hand, the underlying topological cycles will decide the sparsity of the proposed PGO formulation.

C. PGO in Cycle Space

Alternatively, we can traverse edges sequentially along a topological cycle, and consider the associated relative poses, to obtain a geometric cycle. For example, consider a topological cycle with length $\boldsymbol{\lambda}$. Let the sequential relative poses along the cycle be $\mathbf{T}_{\mathbf{1}^c}, \mathbf{T}_{\mathbf{2}^c}, \ldots, \mathbf{T}_{\boldsymbol{\lambda}^c}$, and the corresponding orientations of the edges be $\sigma(\mathbf{1}^c), \sigma(\mathbf{2}^c), \ldots, \sigma(\boldsymbol{\lambda}^c)$, where $\sigma(\mathbf{k}^c)$ takes value $+1$ if the traversal uses the edge $\mathbf{k}^c$ in the forward direction, and -1 otherwise. Then, the corresponding geometric cycle can be written as follows:

$$\boldsymbol{\mathcal{C}}^{\text{lhs}} = \mathbf{I}, \quad \text{with } \boldsymbol{\mathcal{C}}^{\text{lhs}} = \mathbf{T}_{\mathbf{1}^c}^{\sigma(\mathbf{1}^c)}\mathbf{T}_{\mathbf{2}^c}^{\sigma(\mathbf{2}^c)}\cdots\mathbf{T}_{\boldsymbol{\lambda}^c}^{\sigma(\boldsymbol{\lambda}^c)}$$

where $\mathbf{T}^{+1} = \mathbf{T}$ and $\mathbf{T}^{-1} = \mathbf{inv}(\mathbf{T})$, respectively.

Based on the edge orientations and associated relative poses, for each topological cycle $\mathcal{C}$, we can generate a corresponding geometric cycle $\boldsymbol{\mathcal{C}}^{\text{lhs}} = \mathbf{I}$. To characterize the cycle space of the graph $\mathcal{G}$, we need ν independent topological cycles, i.e., a cycle basis of $\mathcal{G}$. Let such a cycle basis be $\boldsymbol{\mathcal{B}} = \{\mathcal{C}_i\}_{i=[1:\nu]}$, and its corresponding cycle matrix be $\mathcal{B}$. Then, given the cycle basis $\boldsymbol{\mathcal{B}}$, we can find ν independent geometric cycles accordingly, denoted as $\{\boldsymbol{\mathcal{C}}_i^{\text{lhs}} = \mathbf{I}\}_{i=[1:\nu]}$.

Then, let us take all $|\mathcal{E}|$ relative poses as new variables to be estimated, and take ν independent geometric cycles as constraints in an optimization problem

$$\begin{aligned}\{\mathbf{T}_\mathbf{k}\}_{\mathbf{k}\in\mathcal{E}} = \arg\min \textstyle\sum_{\mathbf{k}\in\mathcal{E}} \|\mathbf{Log}(\tilde{\mathbf{T}}_\mathbf{k}^{-1}\mathbf{T}_\mathbf{k})\|_{\boldsymbol{\Sigma}_\mathbf{k}}^2 \\ \textbf{s.t.} \quad \boldsymbol{\mathcal{C}}_i^{\text{lhs}} = \mathbf{I} \quad \forall i \in [1:\nu].\end{aligned} \tag{4}$$

This optimization problem has a degree-of-freedom (DOF) $|\mathcal{E}| - \nu = |\mathcal{V}| - 1$, which is the same as the DOF of (2). If an optimal configuration of relative poses is found by solving (4), the objective value becomes minimum and all paths between two vertices in the graph become equivalent (guaranteed by the geometric constraints). Then, a solution to (2) can be calculated by composing the estimates of relative poses along an arbitrary path in the graph (for example, along odometry).

In what follows, we will term the PGO formulation in (4) as CB-PGO, and in contrast, the PGO formulation in (2) as VB-PGO.

D. Solving CB-PGO on Manifold

A typical iterative optimization algorithm on Manifold is driven by a sequence of small perturbations until convergence (to a local optimum). For SE(3), the perturbations are normally applied in the vector space of its Lie algebra, which can be passed to the manifold via the exponential mapping. To solve the PGO formulation in (4), at each iteration t, we would like to find a perturbation $\boldsymbol{\xi}_\mathbf{k}$ for each relative poses $\mathbf{T}_\mathbf{k}$, so that its estimate can evolve from $\hat{\mathbf{T}}_\mathbf{k}^{(t)}$ (estimate at iteration t) to $\hat{\mathbf{T}}_\mathbf{k}^{(t+1)}$ (estimate at iteration $t+1$) as

$$\hat{\mathbf{T}}_\mathbf{k}^{(t+1)} = \hat{\mathbf{T}}_\mathbf{k}^{(t)}\mathbf{Exp}(\boldsymbol{\xi}_\mathbf{k}), \quad \mathbf{k} \in \mathcal{E}.$$

To this end, we linearize the PGO formulation in (4) with respect to the set of perturbations, to a quadratic programming with equality constraints

$$\min \|\mathbf{J}^{-1}\boldsymbol{\xi} + \boldsymbol{\eta}\|_{\boldsymbol{\Sigma}}^2 \qquad \textbf{s.t.} \quad \mathbf{B}\boldsymbol{\xi} + \mathbf{b} = \mathbf{0}. \tag{5}$$

Here, $\mathbf{J}$ and $\boldsymbol{\eta}$ are the Jacobian matrix and the residual vector, respectively, by linearizing the objective function, whose calculations are provided in Appendix A-A. Analogously, $\mathbf{B}$ and $\mathbf{b}$ are the Jacobian matrix, and its corresponding residual vector by linearizing the geometric cycles. The details on how to derive $\mathbf{B}$ and $\mathbf{b}$ can be found in Appendix A-B. Note that the ith block row and $\mathbf{k}$th block column of $\mathbf{B}$ represents the partial derivative of the ith geometric cycle with respect to the $\mathbf{k}$th edge (i.e., relative poses), which is nonzero iff the $\mathbf{k}$th edge is contained in the ith cycle. In other words, the structure of $\mathbf{B}$ is captured by the cycle matrix $\mathcal{B}$.

By letting $\bar{\boldsymbol{\xi}} = \boldsymbol{\Sigma}^{-\frac{1}{2}}(\boldsymbol{\eta} + \mathbf{J}^{-1}\boldsymbol{\xi})$, $\bar{\mathbf{B}} = \mathbf{B}\mathbf{J}\boldsymbol{\Sigma}^{\frac{1}{2}}$, and $\bar{\mathbf{b}} = \mathbf{B}\mathbf{J}\boldsymbol{\eta} - \mathbf{b}$, the quadratic programming in (5) takes the form of a minimum norm optimization problem

$$\min \|\bar{\boldsymbol{\xi}}\|^2 \qquad \textbf{s.t.} \quad \bar{\mathbf{B}}\bar{\boldsymbol{\xi}} = \bar{\mathbf{b}} \tag{6}$$

whose solution is $\bar{\boldsymbol{\xi}}^{\text{opt}} = \bar{\mathbf{B}}^\dagger\bar{\mathbf{b}}$. Note that since both $\boldsymbol{\Sigma}^{\frac{1}{2}}$ and $\mathbf{J}$ are block-diagonal matrices, $\bar{\mathbf{B}}$ and $\mathbf{B}$ would have the same structure. Finally, the perturbation in $\boldsymbol{\xi}$ can be recovered by $\boldsymbol{\xi}^{\text{opt}} = \mathbf{J}(\boldsymbol{\Sigma}^{\frac{1}{2}}\bar{\boldsymbol{\xi}}^{\text{opt}} - \boldsymbol{\eta})$.

The overall algorithm can be termed as SQP [18], [19], and [73], since it requires the solving of a sequence of perturbations via quadratic programming.

Remark 3: The linear system $\bar{\boldsymbol{\xi}}^{\text{opt}} = \bar{\mathbf{B}}^\dagger\bar{\mathbf{b}}$ to be solved in CB-PGO has exactly the same dimension of that in the cycle space, which is $\nu = |\mathcal{E}| - |\mathcal{V}| + 1$, because the Jacobian is characterized by the cycle matrix $\mathcal{B}$. In contrast, an iterative solver to VB-PGO in (2) solves a linear system with a dimension of $|\mathcal{V}| - 1$, i.e, the dimension of cut space, since its Jacobian is characterized by the incidence matrix [37], [39], and [41]. Given the fact that the cut space and cycle space are orthogonal complementary [63], we conclude from the graph topology perspective that the VB-PGO and CB-PGO are counterparts to one another. Moreover, VB-PGO is a least squares optimization for an overdeterminant system, while CB-PGO is a minimum norm optimization for an underdeterminant system. Mathematically, the least squares optimization and minimum norm optimization are highly correlated [74], where both solutions are compactly written as Moore–Penrose pseudo inverse (i.e., left and right inverse, respectively). This fact further confirms that VB-PGO in (2) and CB-PGO (4) are two sides of the same coin. However, while VB-PGO has reached a mature state, the CB-PGO technique is still rather primitive, because of the hardship of choosing a proper cycle basis.

E. Choices of Cycle Basis for PGO

For the CB-PGO in (4), the structure of the Jacobian matrices $\mathbf{B}$ and $\bar{\mathbf{B}}$ is completely described by a cycle matrix $\mathcal{B}$. Obviously, different choices of cycle bases $\boldsymbol{\mathcal{B}}$ lead to different cycle matrices $\mathcal{B}$, which eventually lead to different structures in $\mathbf{B}$ and $\bar{\mathbf{B}}$.

Therefore, we discuss here the pros and cons of different cycle bases for the PGO formulation in (4). We will conclude the advantage of using a minimum cycle basis (MCB) from the sparsity perspective. The discussions on the convergence behavior will be presented in Section VII-B.

1) Fundamental Cycle Basis (FCB): Given an arbitrary spanning tree of the graph, a cycle can be constructed by one off-tree edge (i.e., chord), and the path on the tree connecting the ends of the edge. The set of cycles corresponding to these ν off-tree edges are independent, called FCB. FCB can be generated cheaply, while it cannot ensure a sparse Jacobian matrix in general [19].

2) Minimum Fundamental Cycle Basis (MFCB): A remedy is to use the MFCB, where the spanning tree is chosen in a way such that the summation of the lengths of the fundamental cycles is minimum. An exact solution to MFCB is proven to be NP-complete [75]. While approximate algorithms can solve MFCB in polynomial time [75]–[77], constraining cycle basis to be fundamental may compromise the sparsity.

3) Minimum Overlap Cycle Basis (MOCB): In light of the fact that the matrix to be factorized has the same structure as $\mathcal{B}\mathcal{B}^T$, the sparseness of the matrix decomposition can be guaranteed by minimizing the number of nonzeros in $\mathcal{B}\mathcal{B}^T$. We name a cycle basis that minimizes $|\mathcal{B}\mathcal{B}^T|_0$ as the MOCB, by the fact that an entry at position (i, j) in $|\mathcal{B}\mathcal{B}^T|_0$ is 0 if and only if the cycle i and j do not share common edges. However, there is no clear way on how to compute an MOCB yet.

4) Minimum Length Cycle Basis (MLCB): An MLCB maximizes the sparsity of $\mathcal{B}$, by minimizing the overall length of cycles in the basis, which may in turn resulting a sparse $\mathcal{B}\mathcal{B}^T$. Different from MFCB, the cycles are not confined to be fundamental, thus yielding an easier problem. MLCB is a special case of the well-known minimum cycle basis (MCB) problem [64], with edge weights set to 1.

According to the discussion mentioned above, *we opt to use MLCB for the CB-PGO to maximize the sparsity*. Since MLCB is a special case of the general MCB, we focus on how to compute MCB in the following Section VI.

VI. Computation of Minimum Length Cycle Basis

In this section, we aim at a complete and concise description of the MCB algorithm used for the CB-PGO. We will follow a hybrid approach that first construct a superset that contains MCB, and then apply an independence test to extract an MCB. We will recall the basics of these concepts for completeness, in particular, the construction of Horton set [78] and isometric set [79], and the state-of-the-art method for independence tests [80], while refer interested reader to the review paper [64] for further reading.

Algorithm 1: Minimum Cycle Basis.

```
$\bar{\mathcal{G}} \leftarrow$ SimplifyGraph ($\mathcal{G}$)   $\triangleright$ Smooth out vertices of degree two
APSP $\leftarrow$ LexDijkstra ($\bar{\mathcal{G}}$)   $\triangleright$ Compute a set of consistent all-pairs-shortest-paths
$\boldsymbol{\mathcal{H}} \leftarrow$ HortonSet (APSP)   $\triangleright$ Construct Horton set implicitly
$\boldsymbol{\mathcal{I}} \leftarrow$ IsometricSet ($\boldsymbol{\mathcal{H}}$)   $\triangleright$ Construct isometric set
$\bar{\boldsymbol{\mathcal{B}}} \leftarrow \emptyset$   $\triangleright$ Initialize MCB for $\bar{\mathcal{G}}$
$\boldsymbol{\mathcal{I}} \leftarrow$ SortAscendingByWeight ($\boldsymbol{\mathcal{I}}$)
// Independence test by support vectors
while $|\bar{\boldsymbol{\mathcal{B}}}| \neq \nu$ $\mathcal{C} \leftarrow$ ExtractMinimumWeightCircuit ($\boldsymbol{\mathcal{I}}$)
  if $\mathcal{C}$ is linearly independent from $\bar{\boldsymbol{\mathcal{B}}}$ $\bar{\boldsymbol{\mathcal{B}}} \leftarrow \bar{\boldsymbol{\mathcal{B}}} \cup \mathcal{C}$
  end if $\boldsymbol{\mathcal{I}} \leftarrow \boldsymbol{\mathcal{I}} \backslash \mathcal{C}$
end while
$\boldsymbol{\mathcal{B}} \leftarrow$ ReconstructMCB ($\bar{\boldsymbol{\mathcal{B}}}$)   $\triangleright$ Reconstruct MCB for $\mathcal{G}$
```

On the present architecture, the computational bottleneck of MCB algorithms is the all-pairs-shortest-paths (APSP) [81], [82]. APSP is required to construct the Horton set, and a consistent APSP for the isometric set. Given the fact that graphs occurred in PGO are sparse graphs with positive integer weights, we propose the following two ideas that can substantially improve the performance of APSP: 1) smoothing out vertices of degree two; 2) using LexDijkstra (in Section VI-F) to compute a consistent APSP that can run in parallel and, thus, is more advantageous than the sequential method in [83].

The overall procedure of the proposed MCB algorithm is summarized in Algorithm 1.

A. Superset of MCB: Horton Set

The study of efficient polynomial time minimum cycle basis algorithms started with Horton's work [78], which builds the connection of shortest paths and a cycle in MCB.

Lemma 1 (see [78]): Let $\mathcal{C}$ be a cycle in a minimum cycle basis $\boldsymbol{\mathcal{B}}$. If u and v are two vertices on $\mathcal{C}$.

then $\mathcal{C}$ must contain one of the shortest paths from u to v.

Another key insight from Horton [78] is that using shortest paths, each cycle $\mathcal{C} \in \boldsymbol{\mathcal{B}}$ can be represented as a vertex-edge pair, called a representation of a cycle. In specific, let x be any vertex in $\mathcal{C}$ ($\mathcal{C} \in \boldsymbol{\mathcal{B}}$), then we can always find an edge e_{uv} such that $\mathcal{C}$ can be expressed as $\mathcal{C} = \mathcal{C}(x, e_{uv}) \triangleq \mathcal{P}_{xu} + \mathcal{P}_{xv} + e_{uv}$, where $\mathcal{P}_{xu}$ and $\mathcal{P}_{xv}$ are shortest paths. Based on this observation, Horton [78] proposed a superset (called Horton set) of MCB using all pairs of vertex-edge combinations. Formally, a Horton set is defined as

$$\boldsymbol{\mathcal{H}} = \{\mathcal{C}(x, e_{uv}) \mid x \in \mathcal{V},\ e_{uv} \in \mathcal{E}\}.$$

Note that there might be several shortest paths between the vertices u and v with exactly the same minimum weight, so the choice of $\mathcal{P}_{uv}$ is not unique in general. Horton [78] proved that if all edges in the graph have nonnegative weights, $\boldsymbol{\mathcal{H}}$ would definitively contain an MCB no matter what shortest path $\mathcal{P}_{uv}$ is chosen for each pair of vertices u and v.

Remark 4: Typically, $\boldsymbol{\mathcal{H}}$ is much larger than $\boldsymbol{\mathcal{B}}$. On the one hand, there are degenerated cases in $\boldsymbol{\mathcal{H}}$ that do not form a simple cycle, for example, if e_{uv} is on $\mathcal{P}_{xu}$ or $\mathcal{P}_{xv}$, or if $\mathcal{P}_{xu}$ and $\mathcal{P}_{xv}$ have vertices other than x in common. However, the degenerated cases can be easily removed. On the other hand, $\boldsymbol{\mathcal{H}}$ is a multiset. By choosing different vertex-edge pairs on $\mathcal{C}$, we obtain different representations of $\mathcal{C}$.

B. Superset of MCB: Isometric Circuits

The construction of $\boldsymbol{\mathcal{H}}$ requires the computation of APSP. In Horton's work [78], APSP is allowed us to be arbitrary, i.e., the shortest path $\mathcal{P}_{uv}$ is selected arbitrarily among all shortest paths from u to v. By intentionally selecting APSP to be consistent, we can identify the duplicates in $\boldsymbol{\mathcal{H}}$, and reduce $\boldsymbol{\mathcal{H}}$ to a much smaller set.

Definition 1 (Consistent APSP): For each shortest path $\mathcal{P}_{uv}$ in APSP, let s and t be two arbitrary vertices lying on $\mathcal{P}_{uv}$, then $\mathcal{P}_{st}$, the selected shortest path from s to t in APSP, is a subgraph of $\mathcal{P}_{uv}$.

A consistent APSP can be computed by a lexicographic method [83], [64] (see Definition 3 and Lemma 4 in Appendix C). Given a consistent APSP, a cycle $\mathcal{C}$ is said to be isometric if for any two vertices u and v on $\mathcal{C}$, the chosen shortest path $\mathcal{P}_{uv}$ in APSP is contained in $\mathcal{C}$ [78], [79]. It can be further verified that a cycle $\mathcal{C}$ is isometric if and only if, for each vertex $x \in \mathcal{C}$, there is a unique edge e_{uv}, such that $\mathcal{C} = \mathcal{C}(x, e_{uv})$, with $\mathcal{P}_{xu}$ and $\mathcal{P}_{xv}$ being in the consistent APSP [79]. Last but not least, the set of all isometric cycles is proved to contain an MCB [64], [79].

In a Horton set $\boldsymbol{\mathcal{H}}$ constructed by a consistent APSP, an isometric cycle $\mathcal{C} \in \boldsymbol{\mathcal{H}}$ would contain exactly $|\mathcal{C}|$ representations, i.e., $|\mathcal{C}|$ duplicates in $\boldsymbol{\mathcal{H}}$. Aiming to eliminate redundant representations, we will use the following Lemma to find all the equivalent representations in the Horton set.

Lemma 2 (see[79]): Let $s_x(y)$ be the first vertex (except x) on the shortest path $\mathcal{P}_{xy}$. For any cycle $\mathcal{C} = \mathcal{C}(x, e_{uv}) \in \boldsymbol{\mathcal{H}}$, with $e_{uv} \notin \mathcal{P}_{xu}$, $e_{uv} \notin \mathcal{P}_{xv}$, and $s_x(u) \neq s_x(v)$.

1) if $x = u$ then $\mathcal{C} = \mathcal{C}(v, e_{uv})$.
2) if $x \neq u$, let $x' = s_x(u)$.
   a) if $x = s_{x'}(v)$ then $\mathcal{C} = \mathcal{C}(x,' e_{uv})$.
   b) if $x \neq s_{x'}(v)$, $u = s_v(x')$ then $\mathcal{C} = \mathcal{C}(v, e_{xx'})$.
   c) if $x \neq s_{x'}(v)$, $u \neq s_v(x')$ then $\mathcal{C}$ is not isometric.

Proof: The proof can be found in [79]. Note that the cases $e_{uv} \in \mathcal{P}_{xu}$, $e_{uv} \in \mathcal{P}_{xv}$, and $s_x(u) = s_x(v)$ create bridges, thus, do not form cycles and need to be excluded. An intuitive explanation of isometric cases is presented in Fig. 2.

In Lemma 2 and Fig. 2, x' is chosen on the path $\mathcal{P}(xu)$. However, we can also chose x' on the path $\mathcal{P}(xv)$, i.e., letting $x' = s_x(v)$. By doing so, we can find another equivalent representation for an isometric circuit $\mathcal{C}$ by Lemma 2.

The connection of different representations can be visualized as a directed graph $\mathcal{G}^\dagger = \mathcal{G}^\dagger(\mathcal{V}^\dagger, \mathcal{E}^\dagger)$, where each vertex in $\mathcal{G}^\dagger$ corresponds to a cycle in the Horton set. If a cycle $\mathcal{C}(x, e_{uv})$ is equivalent to a cycle $\mathcal{C}(y, e_{st})$ by Lemma 2, then an arc is formed from the vertex $\mathcal{C}(x, e_{uv})$ to the vertex $\mathcal{C}(y, e_{st})$ in $\mathcal{G}^\dagger$. It was proved that all representations of an isometric cycle $\mathcal{C}$ exactly correspond to a single connected component in $\mathcal{G}^\dagger$ [79]. The following theorem will greatly improve the efficiency of operations on $\mathcal{G}^\dagger$, in terms of graph storage and searching.

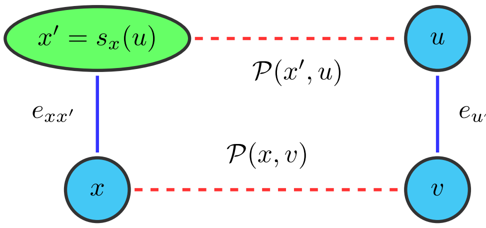


Fig. 2. Illustration of Lemma 2. Let us consider the cycle represented by the vertex x and edge e_{uv}, i.e., $\mathcal{C} = \mathcal{C}(x, e_{uv}) \triangleq \mathcal{P}(x, u) + \mathcal{P}(x, v) + e_{uv}$. $x' = s_x(u)$ is the first vertex on the path $\mathcal{P}(x, u)$, i.e., $\mathcal{P}(x, u) = e_{xx'} + \mathcal{P}(x,' u)$. The key to the proof of Lemma 2 is based on the observation that as follows: If $\mathcal{C}$ is isometric, there are the following two possible cases for the shortest path between the vertex x' and v: (a) $\mathcal{P}(x,' v) = e_{xx'} + \mathcal{P}(x, v)$, (b) $\mathcal{P}(x,' v) = \mathcal{P}(x,' u) + e_{uv}$. Obviously case (a) implies $x = s_{x'}(v)$ and $\mathcal{C} = \mathcal{C}(x,' e_{uv})$, while case (b) implies $u = s_v(x')$ and $\mathcal{C} = \mathcal{C}(v, e_{xx'})$.

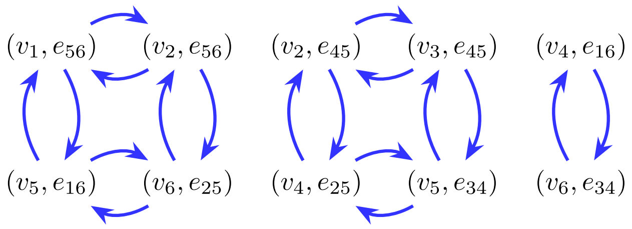


Fig. 3. Visualization of connected components in $\mathcal{G}^\dagger$. Each isometric circuit $\mathcal{C}$ in $\mathcal{G}$ corresponds to a double-linked directed cycle in $\mathcal{G}^\dagger$ with $|\mathcal{C}|$ vertices. This example is created using the graph in Fig. 1, while the cycle representations that do not create links by Lemma 2 are ignored.

Theorem 1: All representations of an isometric circuit $\mathcal{C}$ in $\mathcal{G}^\dagger$ form a double-linked directed cycle with $|\mathcal{C}|$ vertices.

Proof: See Appendix B-A. A visualization of connected components in $\mathcal{G}^\dagger$ is given in Fig. 3.

Based on Theorem 1, the storage of $\mathcal{G}^\dagger$ can be compressed to a vector with two slots reserved for each vertex. We can easily access the adjacent vertices of a given vertex by its index. Besides, a depth-first-search (DFS) [63] on connected components can be simplified as: starting from an arbitrary vertex, keep exploring new vertices until coming back to the start-vertex. This eliminates the use of function recursions or data structures like a stack.

Finally, we characterize the connected components corresponding to isometric cycles by the abovementioned simplified DFS. Duplicate representations of an isometric cycle are then removed by keeping only one representation in the connected component. All representations of nonisometric cycles are discarded. The construction of isometric cycles from a Horton set $\boldsymbol{\mathcal{H}}$ (constructed by a consistent APSP) can be achieved with an amortized complexity $O(|\mathcal{V}||\mathcal{E}|)$ [79].

Remark 5: It should be noted that the set of all isometric cycles is a superset to an MCB, but not all MCBs. In other words, even though APSP is chosen to be consistent, a cycle $\mathcal{C}$ in an arbitrary MCB can be nonisometric [64].

C. Independence Test

To extract an MCB, we sort the set of isometric cycles in nondescending order of weights, and sequentially extract ν linearly independent cycles with the least weights. This procedure is proved to find an MCB [78]. To test the linear independence, we take a circuit as a vector on $\mathbb{Z}_2^m$ incident on the set of edges, and then evaluate the linear independence algebraically.

Given a (spanning) tree $\mathcal{T}$, let the restricted incidence vector of $\mathcal{C}$ be $\bar{\mathcal{C}} = \mathcal{C}\backslash\mathcal{T}$, i.e., considering the off-tree edges of $\mathcal{T}$ only [84]. It can be shown that the linear independence of a collection of cycles $\{\mathcal{C}_i\}_{i\in N}$ implies the linear independence of the corresponding restricted incidence vectors $\{\bar{\mathcal{C}}_i\}_{i\in N}$, and vice versa (see Theorem 3 in Appendix B for a proof).

1) Gaussian Elimination Based Approach: Gaussian elimination is a well-exploited technique in graph theory to check the linear independence of incidence vectors [78], [85]. The basic idea is to stack the set of (restricted) incidence vectors as a matrix, which will be subsequently reduced to the row echelon form. The incidence vectors are linearly independent if and only if the row echelon form has full row rank. This approach has a cubic complexity in the worst case.

2) Support Vector Based Approach: Let $\{\mathcal{C}_i\}_{i=1}^{k-1}$ be a set of independent circuits. Given a spanning tree $\mathcal{T}$, let $\{\bar{\mathcal{C}}_i\}_{i=1}^{k-1}$ be the corresponding restricted incidence vectors whose span is a space $\mathbf{C}_{[1:k-1]}$. Denote the orthogonal complementary space as $\mathbf{S}_{[k:\nu]} = \mathbf{C}_{[1:k-1]}^{\perp}$. Let $\{\bar{\mathcal{S}}_i\}_{j=k}^{\nu}$ be a basis of the space $\mathbf{S}_{[k:\nu]}$. The vectors $\{\bar{\mathcal{S}}_i\}_{j=k}^{\nu}$ are called support vectors of $\{\mathcal{C}_i\}_{i=1}^{k-1}$ [64], [86]. Then, a circuit $\mathcal{C}_k$ is linearly independent from the set of independent circuits $\{\mathcal{C}_i\}_{i=1}^{k-1}$, if and only if there exists an $\bar{\mathcal{S}}_l$ $(k \le l \le \nu)$, such that $\langle \bar{\mathcal{C}}_k, \bar{\mathcal{S}}_l\rangle = 1$ (see Lemma 3 in Appendix B). If such an $\bar{\mathcal{S}}_l$ is found, then $\{\mathcal{C}_i\}_{i=1}^{k} = \{\mathcal{C}_i\}_{i=1}^{k-1} \cup \mathcal{C}_k$ are a set of independent circuits. The support vectors of $\{\mathcal{C}_i\}_{i=1}^{k}$, i.e., a basis of the space $\mathbf{S}_{[k+1:\nu]} = \mathbf{C}_{[1:k]}^{\perp}$ can be obtained by updating $\{\bar{\mathcal{S}}_i\}_{j=k,j\neq l}^{\nu}$ using $\bar{\mathcal{S}}_l$ [64], [86]. Let

$$\bar{\mathcal{S}}_j' = \begin{cases} \bar{\mathcal{S}}_j & \text{if } \langle \bar{\mathcal{C}}_k, \bar{\mathcal{S}}_l\rangle = 0 \\ \bar{\mathcal{S}}_j + \bar{\mathcal{S}}_l & \text{if } \langle \bar{\mathcal{C}}_k, \bar{\mathcal{S}}_l\rangle = 1 \end{cases}$$

then $\{\bar{\mathcal{S}}_j'\}_{j=k,j\neq l}^{\nu}$ is a set of support vectors for $\{\mathcal{C}_i\}_{i=1}^{k}$.

A direct use of support vectors to check independence can be found in [81]. We invert the process in [81] to accommodate our description. Given a spanning tree $\mathcal{T}$, we can initialize ν independent support vectors by ν off-tree edges, where each support vector contains one off-tree edge [81]. Then, at each phase, we evaluate the independence of a new circuit $\mathcal{C}$ by support vectors, which will be subsequently updated if $\mathcal{C}$ is evidenced to be independent by a support vector $\mathcal{S}_l$. Iterate this process until an MCB is found. As in [81], the drawback of this method is that we might need to check many support vectors in order to verify $\langle \mathcal{C}, \mathcal{S}_l\rangle = 1$.

A more sophisticated design is due to Amaldi *et al.* [80]. The approach is based on the idea that if a circuit $\mathcal{C}$ contains edges that are not used by any selected circuits, then this circuit is independent from the selected ones. If this is the case, we can verify the independence of $\mathcal{C}$ without using any support vector. Moreover, the "new" edges in $\mathcal{C}$ can be used to construct new support vectors. Particularly, there is no need to designate a spanning tree $\mathcal{T}$ and initialize a set of independent support vectors at the beginning of the algorithm. The spanning tree $\mathcal{T}$ is built adaptively by greedily including "new" edges without creating a cycle to maximize the sparsity of $\bar{\mathcal{C}}$. Let the new edges in $\mathcal{C}\backslash\mathcal{T}$ be $\mathcal{C}_N = \{e_1, \dots, e_k\}$, then we can identify maximally k independent support vectors, for example, $\mathcal{S}_0 = \{e_1\}$ and $\mathcal{S}_j = \{e_j, e_{j+1}\}$ $(1 \le j \le k-1)$. Obviously, $\langle \mathcal{C}, \mathcal{S}_0\rangle = 1$ and $\langle \mathcal{C}, \mathcal{S}_k\rangle = 0$, which means $\mathcal{S}_0$ is an implicit support vector that evidences $\mathcal{C}$, and there is no need to update $\mathcal{S}_k$ for $\mathcal{C}$. If $\mathcal{C}$ does not contain any new edge, the algorithm checks the existing support vectors by Lemma 3 to verify the independence instead. To speed up the inner product $\langle \mathcal{C}, \mathcal{S}\rangle$, the algorithm maintains $\mathcal{E}_{\mathcal{S}}$ to be the edges used by present support vectors, and $\mathcal{E}^{\circ}$ to be those not anymore because of the update of support vectors ($\mathcal{E}_{\mathcal{S}} \cup \mathcal{E}^{\circ}$ is the set of off-tree edges). At each stage, the edges in both $\mathcal{T}$ and $\mathcal{E}^{\circ}$ can be excluded to increase the sparsity of $\bar{\mathcal{C}}$.

We will use the algorithm by Amaldi *et al.* [80] to extract an MCB from the set of isometric cycles.

D. Smoothing Out Vertices of Degree Two

For PGO, the underlying graph is usually sparse. Furthermore, we assume that the sparsity of PGO is positively correlated to the proportion of vertices of degree two in the graph. The sparser the graph is, the more vertices of degree two we have. The vertices of degree two have no contribution to the topology of the graph, thus, can be pruned out for the computation of an MCB. The pruning of vertices of degree two, along with the edges incident to them, would greatly reduce the combinatorial complexity of the Horton set.

Algorithmically, we can perform a DFS from a node whose degree is not two. During the search, if a vertex of degree two is detected, we greedily probe along the "degree two chain" until a vertex whose degree is not two is found. Then, we replace the "degree two chain" by a new edge (let us name it a "chain edge") whose weight is the accumulated weight along the chain (see Fig. 4). The DFS is recursively called at every unvisited vertex whose degree is not two. Finally, after computing an MCB on the reduced graph, the "chain edges" can be replaced back by the corresponding "degree two chains" to obtain an MCB of the original graph.

Remark 6 (Ear Decomposition): The vertices of degree two can also be pruned out using the ear decomposition [87], which requires the graph to be 2-edge connected. A minimum cycle basis algorithm exploiting ear decomposition is provided in [82]. The algorithm in [82] exploiting the idea of feedback vertices to reduce the Horton set, which is encompassed in the concept of isometric cycles [80]. Nevertheless, the set of isometric cycles are much smaller than the set of cycles exploiting the idea of feedback vertices [80]. Actually, an ear decomposition is closely related to a DFS of the graph [87]. As a result, the task of pruning

TABLE II
COMPUTATIONAL TIME OF LEXDIJKSTRA AND MCB ON A QUAD-CORE CPU

		Consistent-APSP					Isometric set	Independence	Proposed	Michail [81]
		Method in [83]				LexDijkstra				
		Dijkstra	Sorting	Processing	Overall					
MITb	Sequel	1.17e-4	6.23e-5	5.97e-5	2.39e-4	1.25e-4	5.76e-5	7.24e-6		
	Parallel	6.50e-4	-	-	7.72e-4	7.20e-4	4.61e-5	-	8.45e-4	2.69e-3
INTEL_P	Sequel	2.42e-3	5.06e-4	8.66e-4	3.79e-3	2.79e-3	9.01e-4	5.35e-5		
	Parallel	1.78e-3	-	-	3.15e-3	1.31e-3	5.46e-4	-	2.11e-3	2.46e-2
KITTI	Sequel	1.36e-3	6.50e-4	1.41e-3	3.42e-3	2.32e-3	1.13e-3	9.49e-5		
	Parallel	2.63e-3	-	-	4.69e-3	2.74e-3	7.34e-4	-	3.94e-3	fail
Intel	Sequel	3.52e-2	1.26e-2	2.66e-2	7.44e-2	5.83e-2	1.49e-2	1.88e-4		
	Parallel	1.43e-2	-	-	5.35e-2	2.45e-2	8.47e-3	-	3.36e-2	7.95e-2
Manhattan	Sequel	0.495	0.250	0.511	1.26	0.623	0.211	5.66e-4		
	Parallel	0.187	-	-	0.948	0.241	0.105	-	0.348	0.598
Sphere2500	Sequel	0.469	0.202	0.601	1.27	2.03	0.215	2.45e-4		
	Parallel	0.187	-	-	0.990	0.766	0.116	-	0.883	0.309
City10k	Sequel	7.74	3.45	11.1	22.3	11.7	4.69	6.25e-3		
	Parallel	2.88	-	-	17.4	4.53	2.71	-	7.26	8.42
Torus10k	Sequel	8.91	3.64	13.6	26.2	14.3	5.93	1.08e-2		
	Parallel	3.36	-	-	20.6	5.64	3.29	-	8.95	14.3

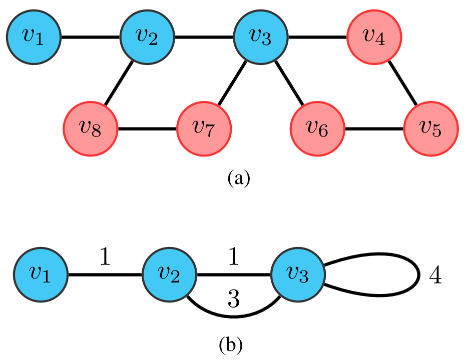


Fig. 4. Original graph (unweighted) and the corresponding reduced graph (weighted) by smoothing out the vertices of degree two. The weight of an edge in the reduced graph is the number of edges it represents in the original graph. Both graphs possess the same cycle structure. (a) Original unweighted graph. (b) Reduced weighted graph.

vertices of degree two can be achieved by DFS explicitly without computing an ear decomposition as an intermediate step.

E. Self-Loops and Multiple-Edges

Sometimes the graph may contain self-loops or multiple-edges, which can be created artificially, or as a consequence of smoothing out vertices of degree two (see Section VI-D and an example in Fig. 4). The self-loops and multiple-edges can be easily coped with by describing paths and cycles as a set of edges (instead of vertices) in the MCB algorithm. It is also possible to eliminate all self-loops and multiple-edges for the MCB computation (see [64, Lemmas 3.17 and 3.18]). However, as shown in the experiments, the computational bottle-neck of the MCB is APSP, while the cost on the independence test is negligible. Thus, we retain self-loops and multiple-edges in the graph, and opt to use edges to describe paths and cycles.

F. LexDijkstra and Parallelism

The bottleneck of the described MCB algorithm is the computation of a consistent APSP (see Table II).

The method described in [83] first compute all-pairs-shortest-distances (by using any shortest paths algorithm), then a set of consistent APSP is constructed by choosing the so-called lexicographic shortest path for each pair of vertices (see Definition 3 and Lemma 4 in Appendix C), by processing vertex-pairs according to distances (i.e., weight and length) from the shortest to the longest. Obviously, a sorting process is required [83], which can be mitigated by a topological sorting [64]. Besides, although not shown in amortized complexity, the random access to shortest path trees is expensive, in particular for a serial processing (by distances).

The algorithm in [83] is general, applicable for graphs with negative weights and any APSP algorithm. However, for a sparse graph with positive weight, Dijkstra [88] is always the preferable shortest paths algorithm. It can be shown that the lexicographic shortest path for each vertex-pairs can be selected by slightly modifying Dijkstra's update process. The resultant APSP is to run Dijkstra for each vertex, which can be easily parallelized by using a multicore CPU. We refer to this consistent APSP method as LexDijkstra.

Proposition 1 (LexDijkstra): Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be an undirected graph with weight $w(e) > 0, \forall e \in \mathcal{E}$. A consistent shortest path $\mathcal{P}_{uv}$ can be obtained for each pair of nodes u, v by modifying Dijkstra's update process to choose the lexicographic path in Definition 3 (provided in Appendix C).

Proof: See Appendix C-A for the proof, and Theorem 4 in Appendix C that supports this result. ■

In terms of the lexicographic comparison defined in Definition 3, cases (1) and (2) are rather cheap. However, case (3) in the worst case, needs to traverse and compare all the edges in $\mathcal{P}'_{uv}$ and $\mathcal{P}'_{uv}$, which is rather inefficient. To address this issue, we propose the following theorem that greatly reduces the complexity in case (3).

Theorem 2: Let $\mathcal{P}_{uv}$ and $\mathcal{P}'_{uv}$ be two paths from u to v. In LexDijkstra, if the algorithm reaches the case (3) of Definition 3, it just suffices that the algorithm traverses back to the nearest common vertex that shared by $\mathcal{P}_{uv}$ and $\mathcal{P}'_{uv}$.

Proof: See Appendix C-B. ■

For the construction of isometric circuits, the algorithm requires random access to the shortest path trees. However, this part can be parallelized without any additional effort.

G. Complexity

The overall computational time of the proposed MCB algorithm is presented in Table II, running on a quad-core CPU. Table II shows that the time spending on the independence test is negligible compared with that spending on computing a consistent APSP and constructing the isometric set. Let $\bar{\mathcal{G}}(\bar{\mathcal{V}}, \bar{\mathcal{E}})$ be the reduced graph. Let $m = |\bar{\mathcal{E}}|$, $n = |\bar{\mathcal{V}}|$.

1) Computing a Consistent APSP: Let us compare the proposed LexDijkstra with the Method in [83]. LexDijkstra: The Dijkstra algorithm has a sorting bottleneck, which is usually addressed by a priority queue. In one single Dijkstra run, for each new edge, the operation on the priority queue has a complexity $O(\log n)$. In the worst case, the lexicographic comparison between two paths takes $O(n)$ operations. Therefore, each new edge contributes $O(\log n + n)$ worst case complexity, which results in $O(m(\log n + n))$ operations for one single LexDijkstra run, and $O(nm(\log n + n)$ operations to compute a consistent APSP using LexDijkstra. Method in <xref ref-type="bibr" rid="ref83">[83]</xref>: By the method in [83], the overall operations used to compute a consistent APSP is $O(nm \log n + n^2 \log n + nm)$, where the term $O(nm \log n)$ accounts for operations to compute an arbitrary APSP based on the Dijkstra algorithm, $O(n^2 \log n)$ for sorting paths according to weights and lengths, and $O(nm)$ for constructing the consistent shortest paths.

By the worst cases complexity, it seems that LexDijkstra does not offer any benefits compared with the method in [83]. However, this is not the cases in practice (see Table II). The reason is four folds: First, $O(nm)$ in the method [83] cannot be eliminated because it requires random access to APSP trees, which is expensive on modern architectures. Second, the method in [83] cannot be run in full parallelism since the sorting term $O(n^2 \log n)$ and processing term $O(nm)$ are sequential operations. Third, $O(n)$ is the worst case complexity for lexicographic comparison, while in practice the operations can be greatly reduced due to the inherent asymmetry in the graph and by Theorem 2 as well. Last but not least, LexDijkstra can run in full parallelism, which can take advantage of multicore CPU architectures.

2) Constructing the Isometric Set: It takes $O(nm)$ operations to find a single representation for each isometric circuit [79]. However, there is a big constant due to the random access to the APSP trees, which is implemented as a dense $n \times n$ matrix. While the running time of this part is not major in Table II compared with the expenses on the consistent APSP, we believe this part can be further improved using a sparse storage for APSP trees, given that most of the elements in the dense matrix are redundant because of the consistency of shortest paths.

VII. Discussions

A. Observability

We propose to define the observability in the MLE as follows.

Definition 2 (Observability of MLE [89]): Let $\mathcal{M}^d$ be a manifold of dimension d. A noise-free system $\mathbf{Z} = \mathfrak{F}(\mathbf{X}) : \mathcal{M}_1^{d_1} \rightarrowtail \mathcal{M}_2^{d_2}$, is locally observable at $\mathbf{X}_0 \in \mathcal{M}_1^{d_1}$ if there is a neighborhood of $\mathbf{X}_0$, denoted by $\mathbb{U}_{\mathbf{X}_0}$, such that

$$\forall \mathbf{X} \in \mathbb{U}_{\mathbf{X}_0},\ \mathbf{X} \neq \mathbf{X}_0,\ \text{we have } \mathfrak{F}(\mathbf{X}) \neq \mathfrak{F}(\mathbf{X}_0).$$

By Definition 2, it is easy to verify that VB-PGO in (2) is unobservable. Because given a solution $\mathbf{X}_0 \triangleq \{\mathbf{T}_i\}_{i \in \mathcal{V}}$ and any neighborhood $\mathbb{U}_{\mathbf{X}_0}$ around $\mathbf{X}_0$, by the fact that a Lie group is a continuous group, we can always find a shifted solution $\mathbf{X} \triangleq \{\mathbf{T}'\mathbf{T}_i\}_{i \in \mathcal{V}}, \mathbf{X} \in \mathbb{U}_{\mathbf{X}_0}$, which yields exactly the same measurements (see Remark 2).

This result coincides with the FIM-based observability tool (see Remark 7).

Now, we extend Definition 2 to estimation problems with constraints, i.e., a noise-free sytem $\mathbf{Z} = \mathfrak{F}(\mathbf{X}) : \mathcal{M}_1^{d_1} \rightarrowtail \mathcal{M}_2^{d_2}$ with constraints $\mathfrak{G}(\mathbf{X}) = \mathbf{I} : \mathcal{M}_1^{d_1} \rightarrowtail \mathcal{M}_3^{d_3}$. If the constraint forms a submanifold $\mathcal{M}_4^{d_4}$ embedded on $\mathcal{M}_1^{d_1}$, then we can reduce the constrained MLE to an unconstrained MLE, and verify the observability by Definition 2. The basic tool is the so-called submersion theorem (see [90, Proposition 3.3.3]): If $\mathfrak{G}$ is smooth, and $\mathbf{I}$ is a regular value of $\mathfrak{G}$ (i.e., the rank of $\mathfrak{G}$ is d_3 for each point in $\mathbf{Y} \in \mathcal{M}_1^{d_1}$ satisfying $\mathfrak{G}(\mathbf{Y}) = \mathbf{I}$), then

$$\mathcal{M}_4 = \{\mathbf{X} \mid \mathbf{X} \in \mathcal{M}_1^{d_1},\ \mathfrak{G}(\mathbf{X}) = \mathbf{I},\ \mathbf{rank}(\mathfrak{G}) = d_3\}$$

admits a differential structure, and $\mathcal{M}_4^{d_4}$ is an embedded submanifold of $\mathcal{M}_1^{d_1}$, with dimension $d_4 = d_1 - d_3$. If a set of constraints is "redundant," we can verify a submanifold by the subimmersion theorem (see [90, Proposition 3.3.4]).

Finally let us examine the case of CB-PGO. Let $\mathcal{M}_1^{d_1}$ be $\mathrm{SE}(3)^m$. The ν constraints clearly satisfy the submersion theorem, thus, the constraints admit a submanifold $\mathcal{M}_4^{d_4}$ embedded in $\mathcal{M}_1^{d_1}$. The noise-free system, i.e., the measurement function of relative poses $\mathbf{Z} = \mathfrak{F}(\mathbf{X}) = \mathbf{X}$, with $\mathbf{X} \triangleq \{\mathbf{T_k}\}_{\mathbf{k} \in \mathcal{E}}$, is bijective on $\mathcal{M}_1^{d_1}$, thus, its restriction to $\mathcal{M}_4^{d_4}$ is also bijective. Therefore, taking CB-PGO as an MLE on $\mathcal{M}_4^{d_4}$, we verify CB-PGO to be observable by Definition 2.

Remark 7: In estimation theory, a common practice is to define the observability of an estimation problem as the invertibility of the FIM [91], (also see [92], [93] for applications in robotics). While FIM being a statistical tool, which is related to a specific noise model (like Gaussian), the deterministic definition of observability for MLE in Definition 2 has been shown to be equivalent to FIM-based observability if the probability density function has a continuous derivative [89].

Remark 8: The covariance matrix (whose inversion is FIM) of CB-PGO can be computed in closed form (see [19], [94], [95]).

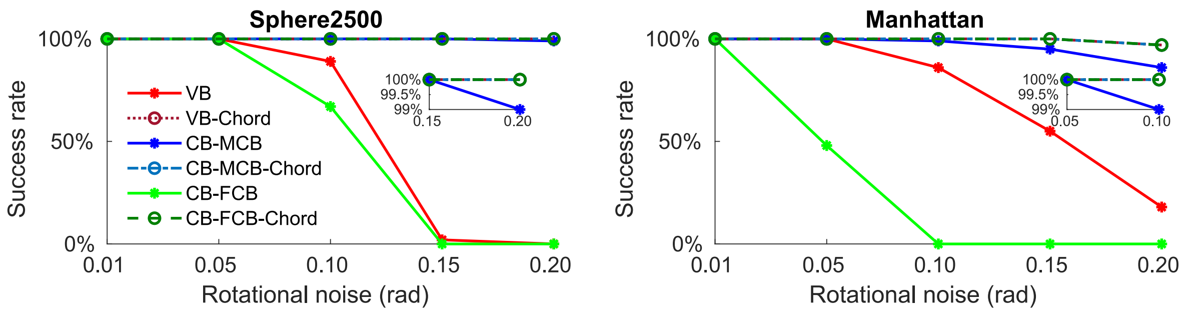


Fig. 5. Robustness of vertex-based approaches and cycle-based approaches on 100 run Monte Carlo simulation. The curves of the chordal bootstrapped methods, i.e., VB-Chord, CB-MCB-Chord, and CB-FCB-Chord, coincide with one other.

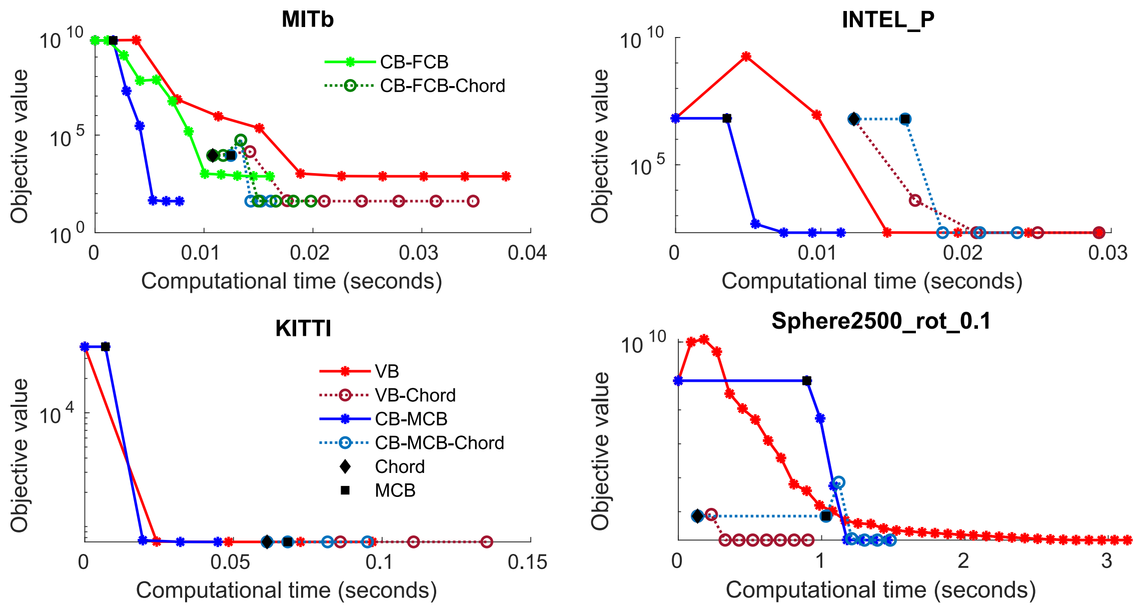


Fig. 6. Convergence of different methods on benchmark datasets. Noet that for INTEL_P, the initial objective values with and without the chordal initialization are 6 281 560 and 6 700 310, respectively, which are close but not the same. Aside from the MCB, the memory usage of VB and CB is comparable. For example, in INTEL_P, the graph storage takes 1.13 MB, while the matrix to be factorized accounts for 8.11 MB; These numbers in CB are 1.34 MB and 8.04 MB, respectively. For MITb, there are 92 nonzero blocks in the Cholesky part of CB-MCB, whereas 2462 for VB. For INTEL_P, CB-MCB accounts for 2234 nonzero blocks, and VB 4194. The numbers for KITTI are 5015 and 13 831, respectively. For dense graphs such as Sphere2500, there are 12 244 nonzero blocks in the Cholesky part of CB-MCB, while the number for VB is 12 398.

However, this covariance matrix is always rank deficient because of the overparameterization in (4), namely $|\mathcal{E}| > \nu$. This does not mean CB-PGO is unobservable. To apply FIM-based tool correctly, we have to obtain FIM for $\mathcal{M}_4^{d_4}$ in its Euclidean space via an atlas [90], instead of using FIM on $\mathcal{M}_1^{d_1}$. Nevertheless, we can verify the observability by Definition 2, without explicitly assigning the atlas.

B. *Jacobian Matrix Design: MCB and Invariance*

In CB-PGO, the entry in the Jacobian matrix [(8) in Appendix XI-B] takes the form of $\sigma(\mathbf{k}^c)\mathbf{Ad}(\boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c))))$, with $\boldsymbol{\mathcal{P}}(\cdot)$ being a geometric path inside the cycle. Let the corresponding topological path be $\mathcal{P}(\cdot)$. Ideally, we want $\mathcal{P}(\cdot)$ to be as short as possible, so that less errors will be accumulated in $\boldsymbol{\mathcal{P}}(\cdot)$, and the Jacobian can more accurately capture the local structure at the linearization point. By using an MCB, the average length of $\mathcal{P}(\cdot)$ is minimized along with the overall length of cycles. Therefore, CB-PGO based on an MCB can be expected to perform better than that based on cycle bases like an FCB. This is true as will be experimentally validated in Figs. 5 and 6.

In CB-PGO, we can minimize the linearization errors inside the Jacobian matrix by using an MCB instead of an FCB. The idea of having a Jacobian matrix less relevant to linearization errors has been exploited in the context of Lie group estimation with "invariance" [96]–[100] as well. In brief, invariance works by choosing a special Lie group parameterization (i.e., group affine [96]) that the Jacobian matrix obtained via linearization at a certain point is merely related to some "error-states" rather than the linearization point directly. As a result the left/right-hand Jacobian in the BCH formula with respect to the error-states is eliminated. This technique can significantly improve the convergence of the estimation problem, especially when facing large noise scenarios.

C. Convergence Rate

Regarding constrained optimization, one of the major concerns is its convergence rate. However, given that CB-PGO in (4) is essentially an equality constrained least squares optimization problem, techniques like SQP can attain quadratic/superlinear convergence [73]. Here, we briefly review some work in this line to confirm the claim. Experimental validations are provided in Fig. 6.

For equality constrained least squares optimization, it has been shown in [101] that a quadratic convergence rate can be obtained by applying the Newton's method (an iterative method to solve a nonlinear equation [102]) to calculate a critical point of the corresponding Lagrangian function. This extends the quadratic convergence of the Newton's method used in unconstrained optimization to constrained cases by SQP. Methods based on approximate Lagrangian Hessians can attain superlinear convergence [103]–[105].

In the context of least squares, the Gauss–Newton method yields quadratic convergence if the residual error at the minimum is zero [22]. The same conclusion stands in the case of equality constrained least squares, which was proved in [106]. This convergence result implies that SQP by linearizing the cost function and constraints [like in (5)] can have basically similar convergence as the Gauss–Newton method.

These justifications are mostly true if the algorithm works around a minimum. In practice, if the noise level in relative poses is reasonable, CB-PGO initialized by the measurements of relative poses is close to the ground-truth. Therefore, CB-PGO can have a better convergence (compared with VB-PGO initialized by odometry) because the working point is closer to a minimum and the Hessian matrix is more accurate.

VIII. Implementation Details

The most well-known open-source implementation of an MCB comes from Dimitrios Michail [81], based on the library of efficient data types and algorithms (LEDA) graph library. However, the code is not maintained anymore for recent LEDA versions, or Ubuntu systems. The other competitive implementations, for example, [80], [82], are not available in open-source. In contrast, our implementation is freely available to the research community, and can be easily used in other MCB related problems.

We implement the MCB algorithm described in Section VI from scratch with C++ Standard Library and C++ Standard Template Library (STL) only. For Dijkstra, we use the priority-queue implementation of STL. We use a dense square matrix with backward pointers to parent vertices to describe the APSP trees. For set operations on support vectors and cycles, sorted sparse vectors are used instead of binary search trees, which is faster by our experiments. OpenMP is used to parallelize CPU computations on multiple cores.

Both CB-PGO and VB-PGO are implemented on an open-source graph optimization library, i.e., SLAM++ [14], which provides the basic operations on block matrices and Cholesky factorization. In SLAM++, we use the default block Cholesky based linear solver, and the approximate minimum degree ordering algorithm (AMD) [107] to reduce the fill-in. Both CB-PGO and VB-PGO use the same Lie group implementation, to avoid the impact of latent numerical round-off errors.

For a fair comparison, we implement chordal initialization [108], [109] as an alternative initialization technique, using Block Cholesky factorization. Instead of initializing the rotational part of poses only [109], we re-estimate the translational part as well. This will give an accurate initial objective value for the chordal-based methods. An initialization of relative poses is obtained by recalculating relative poses with pose estimates, which can be used to initialize CB-PGO.

Noting that for a batch algorithm, we only need to run MCB and AMD once, which will be followed by several iterations of linearization, Cholesky factorization, and state updates.

IX. Experimental Results

In this article, the experiments are carried out by a laptop equipped with a quad-core CPU, Intel(R) Core(TM) i5-5300 U CPU @ 2.30 GHz $\times$ 4, running on Ubuntu 16.04 LTS.

We will use four real datasets and four simulated datasets as standard benchmarks. INTEL_P, Intel and MITb are obtained by processing the raw measurements from wheel odometry and laser range finder [38]. Raw data are available at [110]. KITTI is a pose graph generated by the proSLAM framework [5] from the vision benchmark dataset [111]. The four simulated datasets and their creators are as follows: Manhattan [24], City10k [26], Sphere2500 [26], and Torus10000 [26].

We will use the cycle ratio of a graph, defined as $\nu/|\mathcal{E}|$, to benchmark the graph sparsity. In graph simulations, we regard the local minimum computed by using the ground-truth initialization as the global minimum. Let the objective value at the global minimum be $f^\star$. For any computed solution, if its objective value f satisfies $|f/f^\star - 1| < 0.01$, the solution is counted as a success. The number of successes divided by the number of Monte Carlo runs is defined as the success rate. For CB-PGO, at each iteration, we first calculate a pose configuration through odometry using the current estimate of relative-poses. Then, we substitute the pose configuration to the cost function of VB-PGO, and use the obtained cost as the objective value of CB-PGO.

In this section, the VB-PGO initialized by odometry is denoted as VB, while CB-PGO initialized by the measurements of relative poses is denoted as CB. We distinguish CB-PGO techniques based on the MCB and FCB by denoting them as CB-MCB, CB-PGO using FCB (CB-FCB), respectively. Besides the "natural" initialization for VB-PGO and CB-PGO, we use chordal initialization to bootstrap the rotational part [109]. To evaluate the objective value after chordal initialization, we recompute the optimal translational estimate after the rotational initialization. The chordal bootstrapped VB, and CB variants are termed as VB-Chord, CB-MCB-Chord and CB-FCB-Chord, respectively.

The maximal iterations are set to 50, and we stop the algorithm if the perturbation (i.e., state updates) has a norm less than 0.001. For CB-PGO, we also ensure the constraint residual norm to be less than 0.001.

TABLE III
Computational Time of Chordal Initialization, MCB, and Cholesky Factorization on Benchmark Datasets

	$\mathcal{G}$				$\bar{\mathcal{G}}$		Chord. Init.	MCB	Cholesky Per Iter.		
	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{V}\rvert$	ν	$\nu/\lvert\mathcal{E}\rvert$	$\lvert\bar{\mathcal{E}}\rvert$	$\lvert\bar{\mathcal{V}}\rvert$	all	CB-MCB	CB-MCB	CB-FCB	VB
MITb	827	808	20	**2.42%**	60	41	7.32e-3	8.45e-4	6.74e-5	6.12e-5	1.31e-3
INTEL_P	1483	1228	256	**17.3%**	396	141	1.18e-2	2.11e-3	1.04e-3	3.98e-2	2.28e-3
KITTI	5065	4541	525	**10.4%**	654	130	5.98e-2	3.94e-3	2.60e-3	1.43	1.02e-2
Intel	1835	943	893	48.7%	1515	623	1.45e-2	3.36e-2	3.31e-3	0.985	2.75e-3
Manhattan	5453	3500	1954	35.8%	4350	2397	4.43e-2	0.348	8.16e-3	1.31	8.42e-3
Sphere2500	4949	2500	2450	49.5%	4947	2498	0.131	0.883	8.51e-2	0.262	8.20e-2
City10k	20687	10000	10688	51.7%	19528	8841	0.209	7.26	6.78e-2	fail	6.11e-2
Torus10k	22280	10000	12281	55.1%	21542	9262	0.589	8.95	0.319	fail	0.340

A. MCB

The timing statistics of the proposed MCB is reported in Table II. The proposed MCB algorithm consists of the following three major parts: consistent-APSP, isometric set construction (see Section VI-B), and independence test (see Section VI-C). The construction of Horton set (see Section VI-A) requires an APSP, while its reduction to the isometric set requires a consistent-APSP. In Table II, Dijkstra is used to compute an APSP. The consistent-APSP can be computed via the method in [83], or by LexDijkstra (proposed). All the timings regarding these two consistent-APSP algorithms are presented. To benchmark the overall effectiveness of the proposed MCB, we compare with the state-of-the-art open-source implementation in [81]. In Table II, the symbol "-" in the "parallel" row means that the corresponding computation cannot be parallelized.

In Table II, the proposed MCB algorithm is at least as fast as the state-of-the-art in [81]. Actually, the proposed MCB is much faster for sparse graphs (due to the pruning of degree-two vertices), and slightly faster for most dense graphs, except for the Sphere dataset. This is due to the high symmetry of the graph, which results to many lexicographical comparisons. The implementation in [81] fails at the KITTI dataset, because it cannot handle parallel edges. In addition, Table II clearly shows the advantage of using the proposed LexDijkstra to compute a consistent-APSP, since it avoids the sequel bottleneck in [83]. In the worst case, LexDijkstra accounts for 1.5 times the timing of the pure Dijkstra (APSP), which we believe is already close to the lower borderline. Note that this timing will double without Theorem 2.

In Table II, the computation is parallelized by a quad-core CPU; however, these timings can be more advantageous if the CPU has more cores (like an eight-core CPU or more).

B. Standard PGO Benchmarks

1) Statistics of Each Part: We report the computational time for Cholesky factorization, MCB, and Chordal initialization in Table III. The timings for linearization, state updates, and system matrix constructions are ignored, because these operations are rather cheap compared with the Cholesky part. An exception is CB-FCB: The system matrix allocation in this case can be rather expensive by indexing nonzero elements in Jacobian. In case of Manhattan, it takes almost 1 min to construct the system matrix, and fails in case of City10 k and Torus10 k. The size and sparsity of the benchmarks are included as well.

Regarding Cholesky factorization, CB-MCB and VB have a comparable performance, while CB-MCB is (2–3 times) faster on sparse graphs. The Cholesky part of CB-FCB takes around two magnitudes of time compared with that of CB-MCB/VB, except for the MITb dataset (which has a rather small $\nu = 20$, thus, it actually does not matter whether the system matrix is sparse or not). The timing statistics of the Cholesky part indicate that CB-FCB is not a viable approach (two magnitudes slower than VB in almost any cases).

In Table III, the timing of the MCB and Cholesky parts is comparable for sparse graphs, i.e., the four real dataset, MITb, INTEL_P, KITTI, Intel. Given that we only need to run MCB once, followed by multiple Cholesky iterations, CB-MCB can take advantage in this case, in particular for the MITb, INTEL_P and KITTI datasets. For dense (simulated) graphs, such as Sphere, Manhattan, City10 k, and Torus10 k, since the MCB part is too expensive compared with the Cholesky part, it is not a good choice to use CB-MCB (or CB-MCB-Chord) if timing is a critical consideration.

The chordal initialization accounts for roughly 2–3 iterations of the Cholesky time. This is understandable since it solves a linear system of k times larger (with $k = 2$ for 2-D; and $k = 3$ for 3-D). We will use chordal initialization to boost all the methods (termed VB-Chord, CB-MCB-Chord, CB-FCB-Chord) to examine the robustness.

2) Convergence and Complexity: In Fig. 6, we visualize the convergence of VB, VB-Chord, CB-MCB, CB-MCB-Chord, CB-FCB, and CB-FCB-Chord, against their computational time. The time point of the MCB and Chordal initialization is marked out for a clear visualization.

The MTIb dataset shows a case where VB and CB-FCB converge to a local minimum, where CB-MCB converges to the global minimum. The chordal bootstrapped approaches, VB-Chord and CB-FCB-Chord, can converge to the global minimum as well, while taking a longer time. In this case, CB-MCB is both the fastest and robustest approach. The fast convergence of CB-MCB is further validated by a sparse graph, INTEL_P (17.3% sparsity), where it significantly outperforms VB, and VB-Chord. The results for FCB, FCB-Chord are not shown for INTEL_P as the timings are too large to be plotted on the same scale. Similar results are witnessed on the KITTI dataset,

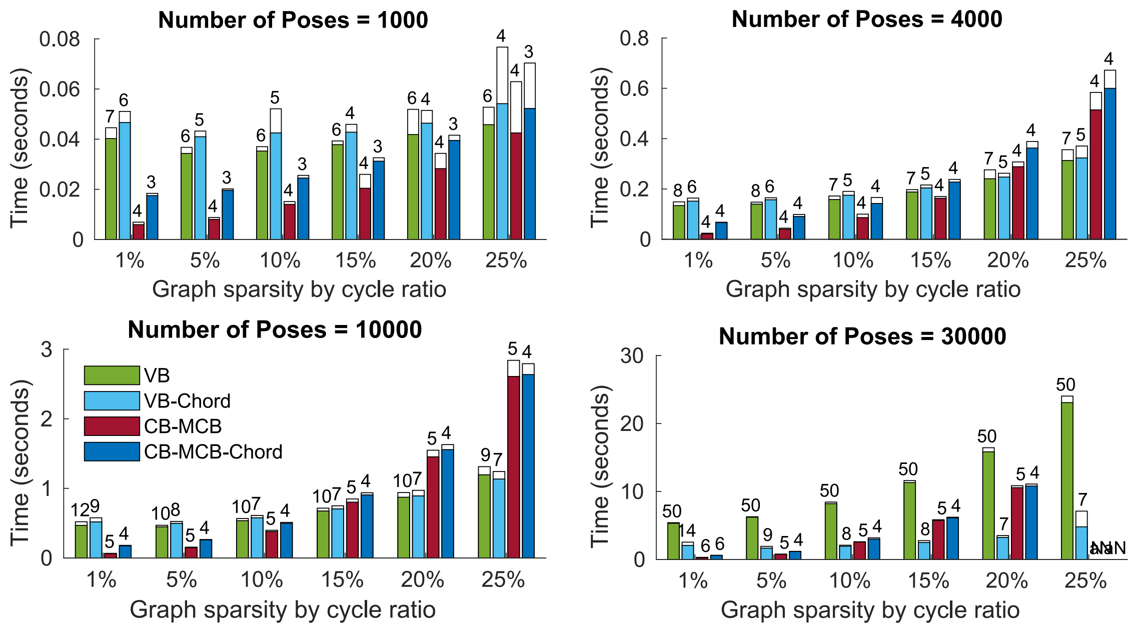


Fig. 7. Computational time of VB (VB-PGO initialized by odometry), VB-Chord (VB-PGO initialized by the Chordal initialization technique), CB-MCB (CB-PGO based on the MCB, initialized by the measurements of relative poses), and CB-MCB-Chord (CB-PGO based on the MCB, initialized by the Chordal initialization technique) on 100 run Monte Carlo simulation. The mean is plotted with color, and the standard deviation is added to the bar-plot as the white margin. The number above each bar is the used iterations. The maximal iteration is set to 50.

while the convergence is much faster given that the dataset is less noisy. For dense graphs, CB-MCB is not advantageous in terms of the computational time, since the MCB part is dominant over the Cholesky part as shown in Table III. However, even in these cases, CB-MCB is still useful because it yields a faster convergence. For instance on the Sphere2500 dataset, if we increase the noise level to 0.1 rads in the rotation part, the convergence of VB degenerates dramatically, and CB-MCB can outperform VB because of the faster convergence. Nonetheless, the VB-Chord can significantly improve the convergence of VB, thus yielding a smaller timing.

C. Monte Carlo Simulation

1) Global Minimum: To examine the robustness of CB-PGO, in terms of converging to the global minimum, we provide a Monte Carlo simulation on the simulated dataset Manhattan (2-D), and Sphere2500 (3-D).

We recreate the dataset from its ground-truth with additive noise in the exponential coordinate, with $0.1\,\mathrm{m}$ standard deviation on the translational part, and 0.01, 0.05, 0.10, 0.15, $0.20\,\mathrm{rads}$, respectively, on the rotational part.

For each case, we generate 100 noisy graphs, and report in Fig. 5, the success rate of VB, VB-Chord, CB-MCB, CB-MCB-Chord, CB-FCB, and CB-FCB-Chord, when solving these graphs. From Fig. 5, we conclude that CB-FCB is basically worse than VB, while CB-MCB is much more robust than VB (initialized by odometry) and CB-FCB. Unsurprisingly, the chordal bootstrapped approaches, i.e., VB-Chord, CB-MCB-Chord, and CB-FCB-Chord show the best robustness. However, a pure CB-MCB (initialized by relative measurements), is almost as robust as the chordal bootstrapped approaches.

2) Computational Time: To better understand the computational complexity of VB, VB-Chord, CB-MCB, and CB-MCB-Chord on sparse graphs, we perform a Monte Carlo simulation, with respect to different cycle ratios and different number of poses. Cases for CB-FCB, CB-FCB-Chord are ignored because the FCB-based approaches do not scale well with respect to graph topologies (see Table III).

We first simulate a giant dense graph using the g2o simulator [13] (g2o_simulator3d). Then, the graph is tailored, respectively, to 1000, 4000, 10 000, and 30 000 poses, and pruned to different sparsity, i.e., 1%, 5%, 10%, 15%, 20%, and 25%. The process randomly generates 100 graphs for each pose number and cycle ratio combination.

The running time statistics of VB, VB-Chord, CB-MCB, and CB-MCB-Chord for each simulated scenario is recorded in Fig. 7, where the mean is plotted with color, and the standard deviation is added to the bar-plot as the white margin. The used iterations are also included above each bar. Note that the maximal iteration is set to 50.

In Fig. 7, CB-MCB is obviously more advantageous than VB for all the tested scenarios with a maximal cycle ratio at around 15%, as well as VB-Chord at round 10%. When the graph becomes denser, with the cycle ratio exceeding 20%, the timings of CB-MCB and CB-MCB-Chord grow with respect to the number of poses as their MCB parts start to dominate the complexity. However, even in these cases, the timings of CB-MCB and CB-MCB-Chord are still comparable to that of VB and VB-Chord, without a dramatical deterioration. The VB-Chord can improve the convergence of VB with a slight tradeoff on the computational cost. The same applies to CB-MCB and CB-MCB-Chord, while the improvement is less obvious. Considering CB-MCB is almost as robust as chordal-based approaches, as shown in Fig. 5, it seems that CB-MCB-Chord does not offer much compared to CB-MCB.

In case of huge graphs, such as 30 000 poses, the numerical stability of VB degenerates dramatically, as can be clearly seen by the iterations consumed. In contrast, the CB-MCB shows a much better numerically stability and convergence property in this scenario, while the bootstrapped approach VB-Chord can significantly improve the convergence of VB. It is worth noting that CB-MCB and CB-MCB-Chord can fail when allocating the memory for shortest path trees (a dense matrix of $O(n^2)$ memory) if the reduced graph still has too many vertices. Such a case is shown at 25% sparsity, with 30 000 poses.

X. Conclusion

To summarize, we propose CB-PGO, a robust and efficient PGO technique that works in the cycle space of the graph. We characterize the graph sparsity by an MCB, which reduces the numerical complexity and enhances the convergence to the global minimum. We design a tailored MCB algorithm for sparse, positive-integer weighted graphs, which can be used for other cycle-based applications as well. The claims on the convergence and computational complexity are validated with experiments. We provide an open-source C++ implementation that is freely available to the community to benefit future research in this direction. The future work includes the following: extending the cycle-based approach to other sparse graph instances; exploiting sparse representations for the consistent APSP storage; developing incremental algorithms for cycle-based approaches; exploring the possibility of using Frobenius norm based cost function and convex relaxations.

Appendix A
Linearization of CB-PGO

In section, we provide the computational details in terms of how to linearize the CB-PGO formulation in (4).

A. Linearization of Cost Function

Denote the error of each edge $\mathbf{k}$ at its estimate $\hat{\mathbf{T}}_\mathbf{k}$ to be $\boldsymbol{\eta}_\mathbf{k} = \mathbf{Log}(\tilde{\mathbf{T}}_\mathbf{k}^{-1} \cdot \hat{\mathbf{T}}_\mathbf{k})$. The linearization of the cost function is trivial using the approximate BCH formula

$$\begin{aligned}
&\sum_{\mathbf{k}\in\mathcal{E}} \|\mathbf{Log}(\tilde{\mathbf{T}}_\mathbf{k}^{-1}\mathbf{T}_\mathbf{k})\|^2_{\boldsymbol{\Sigma}_\mathbf{k}} \\
&\leftarrow \sum_{\mathbf{k}\in\mathcal{E}} \|\mathbf{Log}\left(\tilde{\mathbf{T}}_\mathbf{k}^{-1}\hat{\mathbf{T}}_\mathbf{k}\mathbf{Exp}\left(\boldsymbol{\xi}_\mathbf{k}\right)\right)\|^2_{\boldsymbol{\Sigma}_\mathbf{k}} \\
&= \sum_{\mathbf{k}\in\mathcal{E}} \|\mathbf{Log}\left(\mathbf{Exp}(\boldsymbol{\eta}_\mathbf{k})\mathbf{Exp}(\boldsymbol{\xi}_\mathbf{k})\right)\|^2_{\boldsymbol{\Sigma}_\mathbf{k}} \\
&\approx \sum_{\mathbf{k}\in\mathcal{E}} \|\mathbf{Log}\left(\mathbf{Exp}\left(\boldsymbol{\eta}_\mathbf{k} + \mathbf{J}_\mathbf{r}^{-1}(\boldsymbol{\eta}_\mathbf{k})\boldsymbol{\xi}_\mathbf{k}\right)\right)\|^2_{\boldsymbol{\Sigma}_\mathbf{k}} \\
&= \sum_{\mathbf{k}\in\mathcal{E}} \|\boldsymbol{\eta}_\mathbf{k} + \mathbf{J}_\mathbf{r}^{-1}(\boldsymbol{\eta}_\mathbf{k})\boldsymbol{\xi}_\mathbf{k}\|^2_{\boldsymbol{\Sigma}_\mathbf{k}} = \|\boldsymbol{\eta} + \mathbf{J}^{-1}\boldsymbol{\xi}\|^2_{\boldsymbol{\Sigma}}
\end{aligned}$$

where $\boldsymbol{\eta} = \mathbf{stack}\{\boldsymbol{\eta}_\mathbf{k}\}_{\mathbf{k}\in\mathcal{E}}$, $\boldsymbol{\xi} = \mathbf{stack}\{\boldsymbol{\xi}_\mathbf{k}\}_{\mathbf{k}\in\mathcal{E}}$, $\mathbf{J} = \mathbf{blkdiag}\{\mathbf{J}_\mathbf{r}(\boldsymbol{\eta}_\mathbf{k})\}_{\mathbf{k}\in\mathcal{E}}$, and $\boldsymbol{\Sigma} = \mathbf{blkdiag}\{\boldsymbol{\Sigma}_\mathbf{k}\}_{\mathbf{k}\in\mathcal{E}}$.

B. Linearization of Geometric Cycles

Let us take the metric cycle $\mathcal{C}^{\text{lhs}} = \mathbf{I}$ as an example. Recall that $\mathcal{C}^{\text{lhs}} = \mathbf{T}_{\mathbf{1}^c}^{\sigma(\mathbf{1}^c)}\mathbf{T}_{\mathbf{2}^c}^{\sigma(\mathbf{2}^c)}\cdots\mathbf{T}_{\boldsymbol{\lambda}^c}^{\sigma(\boldsymbol{\lambda}^c)}$, where $\mathbf{T}_{\mathbf{1}^c},\ldots\mathbf{T}_{\boldsymbol{\lambda}^c}$ are the sequential relative poses obtain by a traversal. $\sigma(\mathbf{1}^c),\ldots\sigma(\boldsymbol{\lambda}^c)$ are the relative edge orientations with respect to the traversal, assigned to $+1$ if the traversal uses the edge in the forward direction, and -1 otherwise.

The constraint $\mathcal{C}^{\text{lhs}} = \mathbf{I}$ can be written as $\mathbf{Log}(\mathcal{C}^{\text{lhs}}) = \mathbf{0}$. In the vector space of $\mathfrak{se}(3)$, the first-order Taylor expansion takes the form

$$\sum_{\mathbf{k}^c=\mathbf{1}^c}^{\boldsymbol{\lambda}^c} \frac{\partial\mathbf{Log}(\mathcal{C}^{\text{lhs}})}{\partial\boldsymbol{\xi}_{\mathbf{k}^c}^T}\boldsymbol{\xi}_{\mathbf{k}^c} + \mathbf{Log}(\mathcal{C}^{\text{lhs}}) = \mathbf{0}.$$

At a set of given estimates $\hat{\mathbf{T}}_{\mathbf{k}^c}(\mathbf{k}^c = \mathbf{1}^c\cdots\boldsymbol{\lambda}^c)$, let us define the error of the geometric cycle to be $\boldsymbol{\beta} = \mathbf{Log}(\mathcal{C}^{\text{lhs}})$. For any $\mathbf{h}^c \in [\mathbf{1}^c, \boldsymbol{\lambda}^c]$, with a bit abuse of notation, we split the geometric cycle into two geometric paths $\boldsymbol{\mathcal{P}}(\mathbf{h}^c)$, $\bar{\boldsymbol{\mathcal{P}}}(\mathbf{h}^c)$

$$\begin{aligned}
\boldsymbol{\mathcal{P}}(\mathbf{h}^c) &= \hat{\mathbf{T}}_{\mathbf{1}^c}^{\sigma(\mathbf{1}^c)}\cdots\hat{\mathbf{T}}_{\mathbf{h}^c}^{\sigma(\mathbf{h}^c)} \\
\bar{\boldsymbol{\mathcal{P}}}(\mathbf{h}^c) &= \hat{\mathbf{T}}_{(\mathbf{h+1})^c}^{\sigma((\mathbf{h+1})^c)}\cdots\hat{\mathbf{T}}_{\boldsymbol{\lambda}^c}^{\sigma(\boldsymbol{\lambda}^c)}.
\end{aligned}$$

Let $\boldsymbol{\mathcal{P}}(\mathbf{0}^c) = \mathbf{I}$ be a special case. The equality below is trivial

$$\boldsymbol{\mathcal{P}}(\mathbf{h}^c)\bar{\boldsymbol{\mathcal{P}}}(\mathbf{h}^c) = \mathbf{Exp}(\boldsymbol{\beta}). \tag{7}$$

Now lets add a perturbation $\boldsymbol{\xi}_{\mathbf{k}^c}$ to the edge $\mathbf{k}^c$ inside $\mathcal{C}^{\text{lhs}}$ via the exponential mapping

$$\mathcal{C}^{\text{lhs}} \leftarrow \hat{\mathbf{T}}_{\mathbf{1}^c}^{\sigma(\mathbf{1}^c)}\cdots\left(\hat{\mathbf{T}}_{\mathbf{k}^c}\cdot\mathbf{Exp}(\boldsymbol{\xi}_{\mathbf{k}^c})\right)^{\sigma(\mathbf{k}^c)}\cdots\hat{\mathbf{T}}_{\boldsymbol{\lambda}^c}^{\sigma(\boldsymbol{\lambda}^c)}.$$

Let the perturbed $\mathcal{C}^{\text{lhs}}$ at $\mathbf{k}^c$ be $\mathcal{C}^{\text{lhs}}|_{\mathbf{k}^c}$, which can be compactly written as

$$\mathcal{C}^{\text{lhs}}|_{\mathbf{k}^c} = \boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c)))\mathbf{Exp}\left(\sigma(\mathbf{k}^c)\boldsymbol{\xi}_{\mathbf{k}^c}\right)\bar{\boldsymbol{\mathcal{P}}}(\alpha(\sigma(\mathbf{k}^c)))$$

where $\alpha(\sigma(\mathbf{k}^c))$ is a scalar function with respect to $\sigma(\mathbf{k}^c)$

$$\alpha(\sigma(\mathbf{k}^c)) = \begin{cases}\mathbf{k}^c & \text{if } \sigma(\mathbf{k}^c) = +1 \\ (\mathbf{k}-\mathbf{1})^c & \text{if } \sigma(\mathbf{k}^c) = -1.\end{cases}$$

Applying (1) and considering (7), $\mathcal{C}^{\text{lhs}}|_{\mathbf{k}^c}$ can be written as the multiplication of two matrix exponentials, which can be concatenated by the approximate BCH formula, as

$$\begin{aligned}
\mathcal{C}^{\text{lhs}}|_{\mathbf{k}^c} &= \mathbf{Exp}\left(\sigma(\mathbf{k}^c)\mathbf{Ad}(\boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c))))\boldsymbol{\xi}_{\mathbf{k}^c}\right)\mathbf{Exp}\left(\boldsymbol{\beta}\right) \\
&\approx \mathbf{Exp}\left(\sigma(\mathbf{k}^c)\mathbf{J}_\mathbf{l}^{-1}(\boldsymbol{\beta})\mathbf{Ad}(\boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c))))\boldsymbol{\xi}_{\mathbf{k}^c} + \boldsymbol{\beta}\right).
\end{aligned}$$

At last, the partial derivative could be calculated as follows:

$$\begin{aligned}
\frac{\partial\mathbf{Log}(\mathcal{C}^{\text{lhs}})}{\partial\boldsymbol{\xi}_{\mathbf{k}^c}^T} &= \frac{\partial\mathbf{Log}(\mathcal{C}^{\text{lhs}}|_{\mathbf{k}^c})}{\partial\boldsymbol{\xi}_{\mathbf{k}^c}^T} \\
&= \sigma(\mathbf{k}^c)\mathbf{J}_\mathbf{l}^{-1}(\boldsymbol{\beta})\mathbf{Ad}\left(\boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c)))\right).
\end{aligned}$$

It can be seen that $\mathbf{J}_\mathbf{l}^{-1}(\boldsymbol{\beta})$ occurs for each derivative $\partial\mathbf{Log}(\mathcal{C}^{\text{lhs}}|_{\mathbf{k}^c})/\partial\boldsymbol{\xi}_{\mathbf{k}^c}$, so the final linearized cycle writes

$$\sum_{\mathbf{k}^c=\mathbf{1}^c}^{\boldsymbol{\lambda}^c} \sigma(\mathbf{k}^c)\mathbf{Ad}\left(\boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c)))\right)\boldsymbol{\xi}_{\mathbf{k}^c} + \mathbf{J}_\mathbf{l}(\boldsymbol{\beta})\boldsymbol{\beta} = \mathbf{0}. \tag{8}$$

Without loss of generality, let us assume $\mathcal{C}^{\text{lhs}} = \mathbf{I}$ is the ith geometric cycle. Then, the ith block row and $\mathbf{k}^c$th block column of $\mathbf{B}$ writes $\mathbf{B}_{i,\mathbf{k}^c} = \sigma(\mathbf{k}^c)\mathbf{Ad}(\boldsymbol{\mathcal{P}}(\alpha(\sigma(\mathbf{k}^c))))$. The ith block row of $\mathbf{b}$ writes $\mathbf{b}_i = \mathbf{J}_1(\boldsymbol{\beta})\boldsymbol{\beta}$.

For a set of cycles in a cycle basis, we linearize each one of them and assemble the results in the form of $\mathbf{B}\boldsymbol{\xi} + \mathbf{b} = \mathbf{0}$.

Appendix B Theorems and Proofs on Graph Theory

This section contains proofs of several key conclusions that have been used in the proposed MCB. Section B-A is the proof for Theorem 1 that is used for the construction of an isometric set. Theorem 3 is used for the restriction of cycle/support vectors. Lemma 3 is used in independence tests.

A. Proof of Theorem 1

Proof: It has been proved in [79] that all representations of an isometric cycle $\mathcal{C}$ exactly correspond to a single connected component in $\mathcal{G}^\dagger$. We extend this result in what follows.

There are three possible switches in Lemma 2, i.e., case (1), case (2).(a) and case (2).(b). First, we observe that the switch in each case is mutual: If $\mathcal{C}$ can be switched to $\mathcal{C}'$, then $\mathcal{C}'$ can be switched to $\mathcal{C}$ by the same principle. For case (1), it is straightforward by the fact that $\mathcal{P}_{uv} = \mathcal{P}_{vu}$, so $\mathcal{C}(v, e_{uv}) = \mathcal{C}(u, e_{uv})$. In case (2).(a), we need to prove $\mathcal{C}(x, e_{uv}) = \mathcal{C}(x,' e_{uv})$. Starting from $\mathcal{C}(x,' e_{uv})$, x is the first vertex on the shortest path from x' to v, i.e., $x = s_{x'}(v)$. It can be verified that $x' = s_x(u)$, so according to Lemma 2.2.(a), $\mathcal{C}(x,' e_{uv}) = \mathcal{C}(x, e_{uv})$. In Lemma 2.(2).(b), we prove $\mathcal{C}(x, e_{uv}) = \mathcal{C}(v, e_{xx'})$. Starting from $\mathcal{C}(v, e_{xx'})$, u is the first vertex on the shortest path from v to x', i.e., $u = s_v(x')$. It can be verified that $v \neq s_u(x)$ and $x' = s_x(u)$, so by Lemma 2.(2).(b), $\mathcal{C}(v, e_{xx'}) = \mathcal{C}(x, e_{uv})$. This implies that in $\mathcal{G}^\dagger$, the arcs (i.e., directed edges) are mutual: If there is an arc from $\mathcal{C}$ to $\mathcal{C}'$, then there is an arc from $\mathcal{C}'$ to $\mathcal{C}$ as well.

Second, for an isometric circuit $\mathcal{C}(x, e_{st})$, we can generate exactly two switches because in Lemma 2, the vertex u can be chosen as either s or t, and v as either t or s accordingly. This implies that if a vertex in $\mathcal{G}^\dagger$ is a representation of an isometric circuit, it must have an out-degree $d_{\text{out}} = 2$.

Now assume we replace the two arcs mutually connecting $\mathcal{C}$ and $\mathcal{C}'$ in $\mathcal{G}^\dagger$ by an undirected edge. Considering the fact that all representations of an isometric circuit lie in the same connected component of $\mathcal{G}^\dagger$ [79], we conclude: By replacing mutual arc pairs in $\mathcal{G}^\dagger$ as undirected edges, all representations of an isometric circuit constitute a connected subgraph whose vertices have degree of 2, which is a simple cycle. Therefore, the directed version is a double-linked directed cycle in $\mathcal{G}^\dagger$.

An isometric circuit has $|\mathcal{C}|$ representations, so the directed cycle has $|\mathcal{C}|$ vertices.

Theorem 3: Let $\mathcal{T}$ be any tree (not necessarily a spanning tree) in $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Define $\bar{\mathcal{C}} \in \mathbb{Z}_2^\nu$ to be the incidence vector of $\mathcal{C}$ restricted to the set of off-tree edges in $\mathcal{E}\backslash\mathcal{T}$, i.e.,

$$\mathcal{C} = [\bar{\mathcal{C}}, \tilde{\mathcal{C}}], \text{ where } \bar{\mathcal{C}} \in \mathcal{E}\backslash\mathcal{T},\ \tilde{\mathcal{C}} \in \mathcal{T}.$$

Then, for a collection of cycles $\{\mathcal{C}_i = [\bar{\mathcal{C}}_i, \tilde{\mathcal{C}}_i]\}_{i\in N}$, we have $\mathbf{rank}(\{\mathcal{C}_i\}_{i\in N}) = \mathbf{rank}(\{\bar{\mathcal{C}}_i\}_{i\in N})$.

Proof: Note that $\mathbf{rank}(\{\bar{\mathcal{C}}_i\}_{i\in N}) \leq \mathbf{rank}(\{\mathcal{C}_i\}_{i\in N})$. Then, we verify the inequality shall never be reached. Otherwise, there exist at least a sequence of $\lambda_i (i \in K)$, such that $\sum_{i\in K} \lambda_i \bar{\mathcal{C}}_i = \mathbf{0}$ and $\sum_{i\in K} \lambda_i \tilde{\mathcal{C}}_i \neq \mathbf{0}$. As a result, $\mathcal{C}' = \sum_{i\in K} \lambda_i \mathcal{C}_i$ becomes a tree, containing edges only in $\mathcal{T}$, which is impossible. (By composing cycles, the change of the degree for a vertex is even, which implies that each vertex in $\mathcal{C}'$ has an even degree, thus, $\mathcal{C}'$ cannot be a tree.) ■

Lemma 3: Let $\{\bar{\mathcal{S}}_i\}_{j=k}^{\nu}$ be the support vectors of a collection of independent circuits $\{\mathcal{C}_i\}_{i=1}^{k-1}$.

Then, a circuit $\mathcal{C}_k$ is linearly independent from $\{\mathcal{C}_i\}_{i=1}^{k-1}$, if and only if there exists an $\bar{\mathcal{S}}_l$ $(k \leq l \leq \nu)$ such that $\langle \bar{\mathcal{C}}_k, \bar{\mathcal{S}}_l \rangle = 1$.

Proof: Sufficiency: This is well-known in [64], [84], [86]. Assume $\mathcal{C}_k$ is dependent. Then, there is a nontrivial linear combination $\bar{\mathcal{C}}_k = \sum_{i=1}^{k-1} \lambda_i \bar{\mathcal{C}}_i$. As a result, $\langle \bar{\mathcal{C}}_k, \bar{\mathcal{S}}_j \rangle = \sum_{i=1}^{k-1} \lambda_i \langle \bar{\mathcal{C}}_i, \bar{\mathcal{S}}_j \rangle = 0$ holds for all $k \leq j \leq \nu$, which contradicts the existence of $\bar{\mathcal{S}}_l$. Necessity: Assume $\langle \bar{\mathcal{C}}_k, \bar{\mathcal{S}}_j \rangle = 0$ for all $k \leq j \leq \nu$. Then, the orthogonality $\langle \bar{\mathcal{C}}_i, \bar{\mathcal{S}}_j \rangle = 0$ would hold for all $1 \leq i \leq k$, $k \leq j \leq \nu$. Since $\mathcal{C}_k$ is independent from $\{\mathcal{C}_i\}_{i=1}^{k-1}$, then $\{\mathcal{C}_i\}_{i=1}^{k}$ are a set of independent circuits. By Theorem 3, the restricted circuits $\{\bar{\mathcal{C}}_i\}_{i=1}^{k}$ are also independent. Then, $\{\bar{\mathcal{C}}_i\}_{i=1}^{k} \cup \{\bar{\mathcal{S}}_j\}_{j=k}^{\nu}$ are $\nu + 1$ linearly independent vectors for a space of dimension ν, which cannot be true.

Appendix C Lexicographic Dijkstra (LexDijkstra)

In this section, we provide the proofs related to LexDijkstra. To begin with, lexicographic paths are defined in Definition 3. Lemma 4 builds the connection between lexicographic paths and consistent paths. Theorem 4 is a supportive conclusion to derive LexDijkstra. The correctness of LexDijkstra is proved in Section C-A. Last but not least, Section C-B is the proof for a key result that can greatly improve the effectiveness of lexicographic comparisons.

Definition 3 (Lexicographic Paths [83]): Consider undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with edge weight vector w. Each edge in $\mathcal{E}$ is assigned with a unique index, and a path $\mathcal{P}$ is described as a collection of edges. Then, for every pair of nodes $u, v \in \mathcal{V}$, there exists a unique $u - v$ path $\mathcal{P}_{uv}$ that satisfies exactly one of the following three conditions with respect to any other $u - v$ path $\mathcal{P}'_{uv}$:

1) $w(\mathcal{P}_{uv}) < w(\mathcal{P}'_{uv})$;
2) $w(\mathcal{P}_{uv}) = w(\mathcal{P}'_{uv})$ and $|\mathcal{P}_{uv}| < |\mathcal{P}'_{uv}|$;
3) $w(\mathcal{P}_{uv}) = w(\mathcal{P}'_{uv})$, $|\mathcal{P}_{uv}| = |\mathcal{P}'_{uv}|$, and $\text{min\_index}(\mathcal{P}_{uv}\backslash\mathcal{P}'_{uv}) < \text{min\_index}(\mathcal{P}'_{uv}\backslash\mathcal{P}_{uv})$.

Lemma 4 (see[83]): If all paths in an APSP are lexicographic paths, then the APSP is a consistent APSP.

Proof: The proof is not given in [83], thus, we provide one for completeness.

Let s and t be two vertices on the lexicographic shortest path $\mathcal{P}_{uv}$. Suppose the lexicographic shortest path from s to t, denoted by $\mathcal{P}_{st}$, is not on $\mathcal{P}_{uv}$. Let $\mathcal{P}^*_{st}$ be the subgraph on $\mathcal{P}_{uv}$ joining s and t, i.e., $\mathcal{P}_{uv} = \mathcal{P}_{us} + \mathcal{P}^*_{st} + \mathcal{P}_{tv}$. Let us further define a path $\mathcal{P}'_{uv} = \mathcal{P}_{us} + \mathcal{P}_{st} + \mathcal{P}_{tv}$.

Obviously, $w(\mathcal{P}_{st}^*) = w(\mathcal{P}_{st})$ and $|\mathcal{P}_{st}^*| = |\mathcal{P}_{st}|$ (which implies $w(\mathcal{P}_{uv}) = w(\mathcal{P}'_{uv})$ and $|\mathcal{P}_{uv}| = |\mathcal{P}'_{uv}|$), otherwise either $\mathcal{P}_{uv}$ or $\mathcal{P}_{st}$ is not a lexicographic shortest path by the case (1) or case (2) of Definition 3.

For the case (3) of Definition 3, we observe that

$$\text{min\_index}(\mathcal{P}_{uv} \backslash \mathcal{P}'_{uv}) = \text{min\_index}(\mathcal{P}_{st}^* \backslash \mathcal{P}_{st}) \quad (9)$$

$$\text{min\_index}(\mathcal{P}'_{uv} \backslash \mathcal{P}_{uv}) = \text{min\_index}(\mathcal{P}_{st} \backslash \mathcal{P}_{st}^*). \quad (10)$$

Since $\mathcal{P}_{st}$ is the lexicographic shortest path, we have

$$\text{min\_index}(\mathcal{P}_{st} \backslash \mathcal{P}_{st}^*) < \text{min\_index}(\mathcal{P}_{st}^* \backslash \mathcal{P}_{st}). \quad (11)$$

By (9)–(11), we have $\text{min\_index}(\mathcal{P}'_{uv} \backslash \mathcal{P}_{uv}) < \text{min\_index}(\mathcal{P}_{uv} \backslash \mathcal{P}'_{uv})$. Therefore, by Definition 3.(3), $\mathcal{P}_{uv}$ is not a lexicographic shortest path since $\mathcal{P}'_{uv}$ is lexicographically shorter than $\mathcal{P}_{uv}$, which contradicts the assumption. ■

Theorem 4: Let $\mathcal{G}$ be an undirected graph with weight $w(e) > 0, \forall e \in \mathcal{E}$. A single-source shortest path tree grown at v can be computed using Dijkstra's algorithm [88]. Let $\mathcal{P}_{uv}$ be a shortest path from u to v. Then, for Dijkstra's algorithm, if $w(\mathcal{P}_{uv})$ comes to be the minimum weight available amongst that of all vertices whose shortest path has not been decided yet (such that u is the next vertex to be expanded), all possible paths from u to v with weight $w(\mathcal{P}_{uv})$ have been traversed.

Proof: Let an arbitrary shortest path from u to v with the minimum weight $w(\mathcal{P}'_{uv}) = w(\mathcal{P}_{uv})$ be described by a vertex-edge alternating sequence $\mathcal{P}'_{uv} = \{u - e_{us} - s, \ldots, v\}$. Then, there exists a path $\mathcal{P}_{sv} = \mathcal{P}'_{uv} \backslash e_{us}$ from s to v with weight $w(\mathcal{P}_{sv}) < w(\mathcal{P}_{uv})$ because $w(e_{us}) > 0$. When $w(\mathcal{P}_{uv})$ comes as the minimum weight amongst that of all vertices whose shortest path has not been decided yet, $\mathcal{P}_{sv}$ must have been considered, and a shortest path from s to v must have been found with weight at most $w(\mathcal{P}_{sv})$. Then, at the expansion stage of node s, the edge e_{us} has been traversed, so has been the path $\mathcal{P}'_{uv} = \mathcal{P}_{sv} \cup e_{us}$. ■

A. Proof of Proposition 1

Proof: By Theorem 4, all shortest paths from u to v with exactly the same minimum weight, which is the case (2) and (3) in Definition 3, will be traversed by Dijkstra's algorithm before the final path $\mathcal{P}_{uv}$ is decided. Thus, it would be sufficient to perform a "lexicographic comparison" in Dijkstra's update process according to Definition 3 whenever a new path from u to v is found. By Lemma 4, the APSP comprising lexicographic paths constructed by Definition 3 is consistent. Therefore, $\mathcal{P}_{uv}$ constructed by Definition 3 is a consistent path in a consistent APSP. ■

B. Proof of Theorem 2

Proof: Let $\mathcal{P}_{uv}$ and $\mathcal{P}'_{uv}$ be two paths from u to v.

Let x be an arbitrary common vertex shared by $\mathcal{P}_{uv}$ and $\mathcal{P}'_{uv}$.

Then, $\mathcal{P}_{uv}$ and $\mathcal{P}'_{uv}$ can be divided by x into the following two parts: $\mathcal{P}_{uv} = \mathcal{P}_{ux} + \mathcal{P}_{xv}$, and $\mathcal{P}'_{uv} = \mathcal{P}'_{ux} + \mathcal{P}'_{xv}$. Without loss of generality, let us assume the shortest path tree of LexDijkstra rooted at v. Then, by the time comparing paths between u and v, the lexicographic shortest path between x and v must have been found, and is unique, i.e., $\mathcal{P}'_{xv} = \mathcal{P}_{xv}$. As a result, in case (3) of Definition 3, we have: $\mathcal{P}_{uv} \backslash \mathcal{P}'_{uv} = \mathcal{P}_{ux} \backslash \mathcal{P}'_{ux}$, and $\mathcal{P}'_{uv} \backslash \mathcal{P}_{uv} = \mathcal{P}'_{ux} \backslash \mathcal{P}_{ux}$. Therefore, it suffices to compare the paths (i.e., indices of edges) between u and x. Since x is chosen arbitrary, we compare to the nearest shared vertex. ■

References

[1] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 229–241, Jun. 2001.

[2] 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.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[4] R. Mur-Artal and J. D. Tards, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[5] D. Schlegel, M. Colosi, and G. Grisetti, "PROSLAM: Graph SLAM from a programmer's perspective," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–9.

[6] R. Hartley, J. Trumpf, Y. Dai, and H. Li, "Rotation averaging," *Int. J. Comput. Vis.*, vol. 103, no. 3, pp. 267–305, 2013.

[7] F. Arrigoni, B. Rossi, and A. Fusiello, "Spectral synchronization of multiple views in SE(3)," *SIAM J. Imag. Sci.*, vol. 9, no. 4, pp. 1963–1990, 2016.

[8] R. Tron, X. Zhou, and K. Daniilidis, "A survey on rotation optimization in structure from motion," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2016, pp. 77–85.

[9] S. Esquivel, F. Woelk, and R. Koch, "Calibration of a multi-camera rig from non-overlapping views," in *Proc. Joint Pattern Recognit. Symp.*, 2007, pp. 82–91.

[10] J. Wang, R. K. Ghosh, and S. K. Das, "A survey on sensor localization," *J. Control Theory Appl.*, vol. 8, no. 1, pp. 2–11, 2010.

[11] J. R. Peters, D. Borra, B. Paden, and F. Bullo, "Sensor network localization on the group of three-dimensional displacements," *SIAM J. Control Optim.*, vol. 53, no. 6, pp. 3534–3561, 2015.

[12] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Int. J. Robot. Res.*, vol. 25, no. 12, pp. 1181–1203, 2006.

[13] R. Kmmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 3607–3613.

[14] V. Ila, L. Polok, M. Solony, and P. Svoboda, "SLAM 1-A highly efficient and temporally scalable incremental SLAM framework," *Int. J. Robot. Res.*, vol. 36, no. 2, pp. 210–230, 2017.

[15] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, "SE-Sync: A certifiably correct algorithm for synchronization over the special Euclidean group," *Int. J. Robot. Res.*, vol. 38, no. 2/3, pp. 95–125, 2019.

[16] T. A. Davis, Direct Methods for Sparse Linear Systems, vol. 2. *Philadelphia, PA, USA: Siam*, 2006.

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

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

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

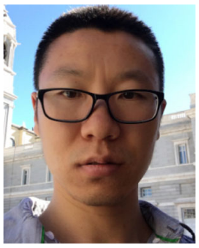
[20] J. Jackson, K. Brink, B. Forsgren, D. Wheeler, and T. McLain, "Direct relative edge optimization, a. robust alternative for pose graph optimization," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1932–1939, Apr. 2019.

[21] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Auton. Robots*, vol. 4, no. 4, pp. 333–349, 1997.

[22] K. Madsen, H. B. Nielsen, and O. Tingleff, *Methods for Non-Linear Least Squares Problems*, 1999. [Online]. Available: http://www2.imm.dtu.dk/pubdb/edoc/imm3215.pdf

[23] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localization and mapping," *IEEE Trans. Robot.*, vol. 21, no. 2, pp. 196–207, Apr. 2005.
[24] 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.
[25] G. Grisetti, C. Stachniss, and W. Burgard, "Nonlinear constraint network optimization for efficient map learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 3, pp. 428–439, Sep. 2009.
[26] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Trans. Robot.*, vol. 24, no. 6, pp. 1365–1378, Dec. 2008.
[27] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Int. J. Robot. Res.*, vol. 31, no. 2, pp. 216–235, 2012.
[28] K. Konolige, "Large-scale map-making," in *Proc. AAAI*, 2004, pp. 457–463.
[29] M. Montemerlo and S. Thrun, "Large-scale robotic 3-d mapping of urban structures," in *Proc. Exp. Robot. IX*, 2006, pp. 141–150.
[30] F. Dellaert, J. Carlson, V. Ila, K. Ni, and C. E. Thorpe, "Subgraph-preconditioned conjugate gradients for large scale SLAM," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 2566–2571.
[31] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963.
[32] D. M. Rosen, M. Kaess, and J. J. Leonard, "RISE: An incremental trust-region method for robust online sparse least-squares estimation," *IEEE Trans. Robot.*, vol. 30, no. 5, pp. 1091–1108, Oct. 2014.
[33] G. Grisetti, R. Kmmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2D and 3D mapping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 273–278.
[34] G. Grisetti, R. Kmmerle, and K. Ni, "Robust optimization of factor graphs by using condensed measurements," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 581–588.
[35] L. Zhao, S. Huang, and G. Dissanayake, "Linear SLAM: A linear solution to the feature-based and pose graph SLAM based on submap joining," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 24–30.
[36] L. Zhao, S. Huang, and G. Dissanayake, "Linear SLAM: Linearising the slam problems using submap joining," *Automatica*, vol. 100, pp. 231–246, 2019.
[37] L. Carlone, R. Aragues, J. A. Castellanos, and B. Bona, "A fast and accurate approximation for planar pose graph optimization," *Int. J. Robot. Res.*, vol. 33, no. 7, pp. 965–987, 2014.
[38] L. Carlone and A. Censi, "From angular manifolds to the integer lattice: Guaranteed orientation estimation with application to pose graph optimization," *IEEE Trans. Robot.*, vol. 30, no. 2, pp. 475–492, Apr. 2014.
[39] K. Khosoussi, S. Huang, and G. Dissanayake, "A sparse separable SLAM back-end," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1536–1549, Dec. 2016.
[40] H. Wang, G. Hu, S. Huang, and G. Dissanayake, "On the structure of nonlinearities in pose graph SLAM," in *Proc. Robot.: Sci. Syst.*, 2013, pp. 425–432.
[41] L. Carlone, "A convergence analysis for pose graph optimization via Gauss-Newton methods," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 965–972.
[42] K. Khosoussi, M. Giamou, G. S. Sukhatme, S. Huang, G. Dissanayake, and J. P. How, "Reliable graph topologies for SLAM," in *Proc. Int. J. Robot. Res.*, vol. 38, no. 2–3, pp. 260–298, 2019.
[43] M. Liu, S. Huang, G. Dissanayake, and H. Wang, "A convex optimization based approach for pose SLAM problems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 1898–1903.
[44] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge, U.K.: Cambridge Univ. Press, 2004.
[45] L. Carlone, G. C. Calafiore, C. Tommolillo, and F. Dellaert, "Planar pose graph optimization: Duality, optimal solutions, and verification," *IEEE Trans. Robot.*, vol. 32, no. 3, pp. 545–565, Jun. 2016.
[46] L. Carlone and F. Dellaert, "Duality-based verification techniques for 2 d SLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 4589–4596.
[47] L. Carlone, D. M. Rosen, G. Calafiore, J. J. Leonard, and F. Dellaert, "Lagrangian duality in 3 d SLAM: Verification techniques and optimal solutions," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 125–132.
[48] D. M. Rosen, C. DuHadway, and J. J. Leonard, "A convex relaxation for approximate global optimization in simultaneous localization and mapping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 5822–5829.
[49] J. Briales and J. Gonzalez-Jimenez, "Cartan-Sync: Fast and global SE(d)-synchronization," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2127–2134, Oct. 2017.
[50] C. Estrada, J. Neira, and J. D. Tards, "Hierarchical SLAM: Real-time accurate mapping of large environments," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 588–596, Aug. 2005.
[51] W. J. Russell, D. J. Klein, and J. P. Hespanha, "Optimal estimation on the graph cycle space," *IEEE Trans. Signal Process.*, vol. 59, no. 6, pp. 2834–2846, Jun. 2011.
[52] E. B. Olson, "Robust and efficient robotic mapping," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Institute of Technology, Cambridge, MA, USA, 2008.
[53] G. Dubbelman and B. Browning, "COP-SLAM: Closed-form online pose-chain optimization for visual SLAM," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1194–1213, Oct. 2015.
[54] C. X. Guo *et al.*, "Large-scale cooperative 3 d visual-inertial mapping in a manhattan world," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 1071–1078.
[55] Z. Wang, S. Huang, and G. Dissanayake, "Decoupling localization and mapping in SLAM using compact relative maps," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 3336–3341.
[56] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," in *Robotics: Science and Systems*, vol. 3. Cambridge, MA, USA: MIT Press, 2007, pp. 65–72.
[57] D. Sibley, C. Mei, I. D. Reid, and P. Newman, "Adaptive relative bundle adjustment," in *Proc. Robot.: Sci. Syst.*, Seattle, USA, vol. 32, Jun. 2009, doi: 10.15607/RSS.2009.V.023.
[58] S. Anderson, K. MacTavish, and T. D. Barfoot, "Relative continuous-time SLAM," *Int. J. Robot. Res.*, vol. 34, no. 12, pp. 1453–1479, 2015.
[59] F.-A. Moreno, J.-L. Blanco, and J. Gonzalez-Jimenez, "A constant-time SLAM back-end in the continuum between global mapping and submapping: Application to visual stereo SLAM," *Int. J. Robot. Res.*, vol. 35, no. 9, pp. 1036–1056, 2016.
[60] G. S. Chirikjian, *Stochastic Models, Information Theory, and Lie Groups*, Volume 2:Analytic Methods and Modern Applications,vol. 2. Berlin, Germany: Springer, 2011.
[61] T. D. Barfoot, *State Estimation for Robotics*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
[62] S. S. Ray, *Graph Theory With Algorithms and Its Applications: In Applied Science and Technology*, Berlin, Germany: Springer, 2012.
[63] T. Kavitha *et al.*, "Cycle bases in graphs characterization, algorithms, complexity, and applications," *Comput. Sci. Rev.*, vol. 3, no. 4, pp. 199–243, 2009.
[64] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," *Proc. SPIE*, vol. 1611, pp. 586–607, 1992.
[65] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *Proc. Robot.: Sci. Syst.*, 2009, vol. 2, no. 4, p. 435.
[66] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge, U.K.: Cambridge Univ. Press, 2003.
[67] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robot. Autom. Mag.*, vol. 18, no. 4, pp. 80–92, Dec. 2011.
[68] M. Cummins and P. Newman, "FAB-MAP: Probabilistic localization and mapping in the space of appearance," *Int. J. Robot. Res.*, vol. 27, no. 6, pp. 647–665, 2008.
[69] S. Lowry *et al.*, "Visual place recognition: A survey," *IEEE Trans. Robot.*, vol. 32, no. 1, pp. 1–19, Feb. 2016.
[70] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, "A certifiably correct algorithm for synchronization over the special Euclidean group," in *Proc. 12th Int. Workshop Algorithmic Foundations Robot.*, 2016.
[71] F. Dellaert *et al.*, "Factor graphs for robot perception," *Found. Trends Robot.*, vol. 6, no. 1/2, pp. 1–139, 2017.
[72] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.
[73] C. D. Meyer, Matrix Analysis and Applied Linear Algebra, vol. 71. Philadelphia, PA, USA: SIAM, 2000.
[74] N. Deo, G. Prabhu, and M. S. Krishnamoorthy, "Algorithms for generating fundamental cycles in a graph," *ACM Trans. Math. Softw.*, vol. 8, no. 1, pp. 26–42, 1982.
[75] E. Amaldi, L. Liberti, F. Maffioli, and N. Maculan, "Edge-swapping algorithms for the minimum fundamental cycle basis problem," *Math. Methods Oper. Res.*, vol. 69, no. 2, pp. 205–233, 2009.
[76] G. Galbiati, R. Rizzi, and E. Amaldi, "On the approximability of the minimum strictly fundamental cycle basis problem," *Discr. Appl. Math.*, vol. 159, no. 4, pp. 187–200, 2011.
[77] 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.

[78] E. Amaldi, C. Iuliano, T. Jurkiewicz, K. Mehlhorn, and R. Rizzi, "Breaking the $\tilde{O}(m^2 n)$ barrier for minimum cycle bases," in *Proc. Eur. Symp. Algorithms.*, 2009, pp. 301–312.
[79] E. Amaldi, C. Iuliano, and R. Rizzi, "Efficient deterministic algorithms for finding a minimum cycle basis in undirected graphs," in *Proc. Int. Conf. Integer Program. Combinatorial Optim.*, 2010, pp. 397–410.
[80] K. Mehlhorn and D. Michail, "Implementing minimum cycle basis algorithms," *J. Exp. Algorithmics*, vol. 11, pp. 2–5, 2007.
[81] D. Dutta, M. Chaitanya, K. Kothapalli, and D. Bera, "Applications of ear decomposition to efficient heterogeneous algorithms for shortest path/cycle problems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2017, pp. 864–873.
[82] D. Hartvigsen and R. Mardon, "The all-pairs min cut problem and the minimum cycle basis problem on planar graphs," *SIAM J. Discr. Math.*, vol. 7, no. 3, pp. 403–418, 1994.
[83] J. de Pina, "Applications of shortest path methods," Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, Dec. 19, 1995. [Online]. Available: https://dare.uva.nl/search?identifier=93573ea1-c3ea-4321-a479-294c74b7f0bb
[84] A. Golynski and J. D. Horton, "A polynomial time algorithm to find the minimum cycle basis of a regular matroid," in *Scandinavian Workshop on Algorithm Theory*. Berlin, Germany: Springer, 2002, pp. 200–209.
[85] T. Kavitha, K. Mehlhorn, D. Michail, and K. E. Paluch, "An $\tilde{O}(m^2 n)$ algorithm for minimum cycle basis of graphs," *Algorithmica*, vol. 52, no. 3, pp. 333–349, 2008.
[86] V. Ramachandran, "Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity," University of Texas at Austin, Austin, TX, USA, Tech. Rep. TX 78712, 1992.
[87] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
[88] C. Jauffret, "Observability and fisher information matrix in nonlinear regression," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 43, no. 2, pp. 756–759, Apr. 2007.
[89] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*, Princeton, NJ, USA: Princeton Univ. Press, 2009.
[90] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation With Applications to Tracking and Navigation: Theory Algorithms and Software*, Hoboken, NJ, USA: Wiley, 2004.
[91] J. Andrade-Cetto and A. Sanfeliu, "The effects of partial observability when building fully correlated maps," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 771–777, Aug. 2005.
[92] Z. Wang and G. Dissanayake, "Observability analysis of SLAM using Fisher information matrix," in *Proc. 10th Int. Conf. Control, Autom., Robot. Vis.*, 2008, pp. 1242–1247.
[93] J. D. Gorman and A. O. Hero, "Lower bounds on parametric estimators with constraints," in *Proc. 4th Annu. ASSP Workshop Spectr. Estimation Model.*, 1988, pp. 223–228.
[94] T. L. Marzetta, "A simple derivation of the constrained multiple parameter Cramer-Rao bound," *IEEE Trans. Signal Process.*, vol. 41, no. 6, pp. 2247–2249, Jun. 1993.
[95] A. Barrau and S. Bonnabel, "Linear Observation Systems on Groups (I)," Feb. 2018. [Online]. Available: https://hal-mines-paristech.archives-ouvertes.fr/hal-01671724
[96] A. Barrau and S. Bonnabel, "Invariant kalman filtering," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, pp. 237–257, 2018.
[97] T. Zhang, K. Wu, J. Song, S. Huang, and G. Dissanayake, "Convergence and consistency analysis for a 3-d Invariant-EKF SLAM," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 733–740, Apr. 2017.
[98] P. Chauchat, A. Barrau, and S. Bonnabel, "Invariant smoothing on lie groups," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1703–1710.
[99] R. A. Tapia, "Quasi-Newton methods for equality constrained optimization: Equivalence of existing methods and a new implementation," in *Nonlinear Programming 3*. New York, NY, USA: Elsevier, 1978, pp. 125–164.
[100] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, vol. 30. Philadelphia, PA, USA: Siam, 1970.
[101] P. T. Boggs, J. W. Tolle, and P. Wang, "On the local convergence of quasi-Newton methods for constrained optimization," *SIAM J. Control Optim.*, vol. 20, no. 2, pp. 161–171, 1982.
[102] R. Fontecilla, T. Steihaug, and R. A. Tapia, "A convergence theory for a class of quasi-Newton methods for constrained optimization," *SIAM J. Numer. Anal.*, vol. 24, no. 5, pp. 1133–1151, 1987.
[103] T. F. Coleman, "On characterizations of superlinear convergence for constrained optimization," *Lectures Appl. Math.*, vol. 26, pp. 113–133, 1990.
[104] H. Schwetlick, "Gauss-Newton-like methods for nonlinear least squares with equality constraints-local convergence and applications," *Statist.: J. Theor. Appl. Statist.*, vol. 16, no. 2, pp. 167–178, 1985.
[105] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, ACM New York, NY, USA, vol. 30, no. 3, pp. 381–388, 2004.
[106] D. Martinec and T. Pajdla, "Robust rotation and translation estimation in multiview reconstruction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–8.
[107] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert, "Initialization techniques for 3 d SLAM: A survey on rotation estimation and its use in pose graph optimization," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 4597–4604.
[108] Pre-2014 Robotics 2D-Laser Datasets. [Online]. Available: http://www.ipb.uni-bonn.de/datasets/
[109] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.



Fang Bai (Member, IEEE) was born in Ningxia Province, China, in 1988. He received the B.Sc. degree in computer science and technology from Nankai University, Tianjin, China, in 2010, and the Ph.D. degree in robotics from the University of Technology Sydney, Ultimo, NSW, Australia, in 2020.

His Ph.D. thesis studied theoretical aspects in graph optimization, resulting a cycle based approach, and a closed form equation to predicting the optimal values in least squares optimization. His research interests include both mathematical abstractions and practical applications in robotics.

Teresa Vidal-Calleja (Member, IEEE) received the B.Sc. degree in mechanical engineering from the National Autonomous University of Mexico (UNAM), Mexico City, Mexico, in 2000, the M.Sc. degree in electrical engineering from CINVESTAV-IPN, Mexico City, Mexico, in 2002, and the Ph.D. degree in automatic control, computer vision and robotics from the Polytechnic University of Catalonia (UPC), Barcelona, Spain, in 2007.

She was a Postdoctoral Research Fellow with the LAAS-CNRS, Toulouse, France, and the Australian Centre for Field Robotics, University of Sydney, Sydney, NSW, Australia. In 2012, she joined the Centre for Autonomous Systems, University of Technology Sydney (UTS), Ultimo NSW, Australia, where he is currently an Associate Professor. Her research interests are in robotic probabilistic perception.

Giorgio Grisetti (Member, IEEE) received the Ph.D. degree in computer engineering from the Sapienza University of Rome, Rome, Italy, in 2006.

He is currently an Associate Professor with the Sapienza University of Rome. He conducted the Postdoctoral Research with the Autonomous Intelligent Systems Lab, University of Freiburg, Breisgau, Germany, from 2006 to 2010. His research interests lie in SLAM, state estimation and navigation for mobile robots. He is author of more than 100 peer reviewed papers in journals and conferences, and he is known for his contribution to open-source packages addressing SLAM related problems such as GMapping, g2o, normal iterative closest point (NICP), Hog-Man, hamming distance embedding binary search tree (HBST), and Pro-SLAM.