Efficient Decision Making under Uncertainty in High-Dimensional State Spaces

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Khen Elimelech

Submitted to the Senate of the Technion – Israel Institute of Technology
Tammuz 5781, Haifa, June 2021

The Research Thesis was Done under the Supervision of Associate Professor Vadim Indelman
in the Technion Autonomous Systems Program

The Generous Financial Help of the Technion is Gratefully Acknowledged

Acknowledgements

First, I would like to thank my supervisor, Professor Vadim Indelman, for his guidance throughout this journey. His continuous advice, immense knowledge, and high standards, were crucial to my growth as a researcher.

I would also like to thank the Technion Autonomous Systems Program (TASP), and the Autonomous Navigation and Perception Lab (ANPL), including all their student and staff members, for providing a supportive environment.

Finally, but most importantly, I would like to thank my parents. Without them, none of this would have been possible.

Author's Publications

Journal Articles

[1] K. Elimelech and V. Indelman, "Efficient belief space planning in high-dimensional state spaces using PIVOT: Predictive Incremental Variable Ordering Tactic," *International Journal of Robotics Research (IJRR)*, May 2021, Submitted by invitation.

[2] ——, "Efficient modification of the upper triangular square root matrix on variable reordering," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 675–682, Apr. 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3048663.

[3] ——, "Simplified decision making in the belief space using belief sparsification," *International Journal of Robotics Research (IJRR)*, Dec. 2018, Conditionally accepted.

In Conference Proceedings

[1] K. Elimelech and V. Indelman, "Introducing PIVOT: Predictive Incremental Variable Ordering Tactic for efficient belief space planning," in *International Symposium on Robotics Research (ISRR)*, Vietnam, Oct. 2019.

[2] ——, "Fast action elimination for efficient decision making and belief space planning using bounded approximations," in *International Symposium on Robotics Research (ISRR)*, Chile, Dec. 2017.

[3] ——, "Scalable sparsification for efficient decision making under uncertainty in high dimensional state spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Canada, Sep. 2017, pp. 5668–5673. DOI: 10.1109/IROS.2017.8206456.

[4] ——, "Consistent sparsification for efficient decision making under uncertainty in high dimensional state spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 3786–3791. DOI: 10.1109/ICRA.2017.7989437.

In Collections

[1] K. Elimelech and V. Indelman, "Fast action elimination for efficient decision making and belief space planning using bounded approximations," in *Robotics Research*, ser. Springer Proceedings in Advanced Robotics (SPAR), vol. 10, Springer International Publishing, 2020, pp. 843–858, ISBN: 978-3-030-28619-4. DOI: 10.1007/978-3-030-28619-4_58.

In Professional Workshops

[1] K. Elimelech and V. Indelman, "PIVOT: Predictive incremental variable ordering tactic for efficient belief space planning," in *Workshop on Toward Online Optimal Control of Dynamic Robots, in conjunction with the IEEE International Conference on Robotics and Automation (ICRA)*, Canada, May 2019.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

The fundamental goal of artificial intelligence and robotics research is to allow agents and robots to autonomously plan and execute their actions. To achieve reliable and robust operation, these agents must account for uncertainty; this may derive from dynamic environments, noisy sensors, or inaccurate delivery of actions. Practically, these settings require reasoning over probabilistic states, known as "beliefs". At large, the focus of this work is to allow computationally efficient decision making under uncertainty, which we formulate as decision making in the belief space. To determine the optimal course of action, the agent should predict the future development of its belief, considering multiple candidate actions. Examining beliefs over high-dimensional state vectors (e.g., entire trajectories) can significantly improve estimation accuracy. Yet, it also holds high computational complexity, which makes the real-time solution of this problem extremely challenging. Inherently, solving a decision problem requires evaluation of the candidate actions according to some objective function. In this work, we claim that we can often spare this exhaustive calculation, and, instead, identify and solve a simplified decision problem, which leads to the same (or similar) action selection. A problem may be simplified by adapting each of its components – initial belief, objective function, and candidate actions – such that the objective calculation becomes "easier". In support of this concept, we present several contributions. We begin by presenting a novel and fundamental theoretical framework, which allows to formally analyze candidate optimality, and quantify the loss induced from possible sub-optimal action selection. This analysis approach also allows us to identify actions which are guaranteed to be sub-optimal, and can be eliminated. We then show how the idea of problem simplification can be applied to decision making in the belief space. Practically, the belief is often represented with a graphical model, or with the high-dimensional upper triangular "square root matrix". Thus, we suggest to simplify the problem by considering a sparse approximation of the initial belief (graph or matrix), which can be efficiently updated, in order to calculate the candidates' objective values. Accordingly, we present a scalable belief sparsification algorithm, and analyze its effect on the decision problem using our suggested framework. We follow with another method for reducing the computational complexity of the problem, by optimizing the order of variables in the state vector before planning. This operation can help reducing the number of variables which would be affected by the belief updates. We call this approach PIVOT: Predictive Incremental Variable Ordering Tactic. Unfortunately, changing the order of variables modifies the matrix in a non-trivial way, and implies its costly complete recalculation. Thus, in support of PIVOT, we also present an efficient and parallelizable algorithm for modification of this matrix on variable reordering, by manipulating it directly. We demonstrate the benefits of our methods to the solution of active Simultaneous Localization and Mapping (SLAM) problems for mobile robots, where we manage to significantly reduce computation time, with practically no loss in the quality of solution. Other relevant tasks include autonomous navigation, sensor placement, and robotic manipulation.

Glossary

Table 1: Glossary.

Notation	Definition
$\mathcal{P} \doteq (\boldsymbol{\xi}, \mathcal{A}, V)$	A decision problem, where $\boldsymbol{\xi}$ is the initial state, $\mathcal{A}$ is a set of candidate actions, and V is the objective (value) function.
$\mathcal{P}_s$	A simplified decision problem.
$loss(\mathcal{P}, \mathcal{P}_s)$	The simplification loss.
$\Delta(\mathcal{P}, \mathcal{P}_s)$ / $\delta(\mathcal{P}, \mathcal{P}_s, a)$	The simplification offset / of an action a.
$\mathcal{P}_1 \simeq \mathcal{P}_2$	Action consistency (a relation between decision problems).
x_k	The agent's pose at time-step k.
$\mathcal{X}_k$	The set of state variables at time-step k.
$\boldsymbol{X}_k$	The state vector at time-step k (given a certain variable order).
$\boldsymbol{X}_k^*$	The MAP estimate of $\boldsymbol{X}_k$.
$n_k \doteq \lvert\mathcal{X}_k\rvert$	The state size at time-step k.
u_k	A control action taken from pose x_k.
$\mathcal{U}_{k:l}$	The set of control action taken between time-steps k and l.
$\mathcal{X}_k^o$	The subset of state variables observed from pose x_k.
z_k	An observation taken from pose x_k.
$\mathcal{Z}_{k:l}$	The set of observations taken between time-steps k and l.
$b_k(\mathcal{X}_k) \doteq \mathbb{P}(\mathcal{X}_k \mid \mathcal{U}_k, \mathcal{Z}_k)$	The agent's belief over its state at time-step k.
$FG_k \doteq (\mathcal{X}_k \cup \mathcal{F}_k, \mathcal{E}_k)$	A factor graph representing the belief b_k.
FG_{k+1}^{δ}	A factor graph representing the update to the belief (i.e., new factors and state variables) when moving from time-step k to time-step $k+1$.
$BN_k \doteq (\mathcal{X}_k, \mathcal{D}_k)$	A Bayesian network ("Bayes net") resulting from performing variable elimination on FG_k (given a certain variable order).
$\mathcal{I}nv_{\mathcal{X}}(H)$	The subset of variables $\subseteq \mathcal{X}$, which are involved in the hypothesis H.
$\neg\mathcal{I}nv_{\mathcal{X}}(H)$	The subset of variables $\subseteq \mathcal{X}$, which are uninvolved in the hypothesis H.
$\boldsymbol{J}_k$	The "Jacobian matrix" of belief b_k, containing the linearized Gaussian factors as its rows.
$\boldsymbol{J}_{k+1}^{\delta}$	The "Jacobian matrix" of the update, containing the linearized Gaussian factors added to the belief when moving from time-step k to time-step $k+1$.
$\boldsymbol{\Lambda}_k \doteq \boldsymbol{J}_k^T \boldsymbol{J}_k$	The information matrix of b_k.
$\boldsymbol{R}_k$	The square root matrix of the information matrix at time-step k, such that $\boldsymbol{\Lambda}_k \doteq \boldsymbol{R}_k^T \boldsymbol{R}_k$.
$\mathtt{I}_{\boldsymbol{M}}\{j_1 : j_2\}$	The set of indices of rows of $\boldsymbol{M}$, in which the leading non-zero is between the j_1-th and j_2-th columns.
NZ	Non-Zero.
π	A policy.
Π	A set of (candidate) policies.
$H_{k:l}$	A hypothesis – an assignment to the Markov chain $[u_k, z_k, \ldots, u_l, z_l]$.
ML	Maximum Likelihood (observations).
$\bar{V}$	An information-theoretic objective function (expected information gain).
$\boldsymbol{I_n}$	The identity matrix of size n.
$i_1 : i_2$	The vector $[i_1, ..., i_2]$.
end	The last index.
`qr`	QR factorization.
`chol`	Cholesky factorization.
`blkdiag`$\{\times, \ldots, \times\}$	Returns a matrix with the given inputs on its block diagonal.
`C/COLAMD`	The respective fill-reducing variable order.
$\boldsymbol{M}(\mathcal{I}, \mathcal{J})$	The sub-matrix of elements $(i, j) : i \in \mathcal{I}, j \in \mathcal{J}$ of $\boldsymbol{M}$; $\boldsymbol{M_1 M_2}(\times, \times)$ marks the block of $\boldsymbol{M_1 M_2}$, and $\boldsymbol{M_1} \cdot \boldsymbol{M_2}(\times, \times)$ marks the block of $\boldsymbol{M_2}$.

Chapter 1

Introduction

1.1 Motivation

In this era, intelligent autonomous agents and robots can be found all around us. They are designed for various functions, such as exploring remote domains, e.g., underwater and space; imitating humans and interacting with them; performing repetitive tasks; and ensuring safety of operations. They might be physically noticeable, e.g., personal-use drones, industrial robotic arms, or space rovers; or less so, with the popularization of internet of things (IoT), smart homes, and virtual assistants. Yet, the increasing demand for these "smart" systems presents new challenges, as the integration of robotic agents into everyday life requires them to operate in real time, using inexpensive hardware.

In addition, when deployed in the real world, such agents and robots must be able to robustly handle uncertainty, in order to achieve reliable performance. For example, when solving tasks such as localization, mapping, or manipulation, mobile robots should often infer their state in the environment, while accounting for their inherent noisy actuation, and only relying only on noisy sensing or partial observability. To allow long-term operation, they should also be able to efficiently propagate their uncertainty-aware (and often) high-dimensional state representation through time. This fundamental problem, known as sequential state estimation or inference, has long been a major interest of the robotics and Artificial Intelligence (AI) research communities; hence, over the years, various probabilistic techniques were suggested for its solution, as we shall cover. An important instance of this problem is Simultaneous Localization And Mapping (SLAM) (Cadena et al., 2016), in which we wish to estimate an agent's location, while mapping its surroundings, *in real time*.

The other important concern in allowing autonomous operation is the task of planning and decision making under uncertainty. We may view state estimation as the "passive" part of the problem, in which we are given a stream of information, and are asked to provide an estimate (and possibly a confidence level) of a measure of interest. In contrast, planning may be viewed as the "active" part of the problem: given the state estimation solution (the state estimate or uncertainty-aware representation), we should determine the *future* course of action for the agent, in order to achieve a certain objective. To make a reliable decision, which can be executed safely, we should predict the propagation of uncertainty into the future, considering different candidate actions or policies, and possible hypotheses for the outcomes of actions, future observations, and the environment. Thus, while state estimation might be challenging for real-time solution, planning and decision making are exponentially more demanding.

At large, **the focus of this work is to allow computationally efficient decision making under uncertainty**, which we will formulate as *decision making in the belief space*. Relevant tasks include active-SLAM for mobile robots, autonomous navigation, sensor placement and active sensing, robotic manipulation, and even cognitive tasks, such as dialogue management.

1.2 Background

Before formulating the decision making problem and its solutions, it is important to first discuss the state inference problem, on which it relies.

1.2.1 Sequential Probabilistic State Inference

In this problem, as mentioned, given a set of probabilistic or noisy constraints or measurements (also known as factors), we would like to find the best, i.e., the maximum-a-posteriori (MAP), state estimate. In the context of mobile robotics, this state typically refers to the robot's trajectory of poses, and/or properties of elements in its environment. To be able to express our confidence in the current estimate, and to allow fusion of information arriving from different sources, we typically maintain a *belief* – a posterior distribution over the state, given the available information – rather than a single point estimate. By utilizing Bayes' rule, we are able to update ("propagate") the belief through time-steps, at the arrival of new information, in what is known as "Bayesian Inference". To clear any possible confusion, note that when using the term "belief propagation", we mean "to update the belief through time-steps", and do not refer to the "belief propagation algorithm" (Yedidia et al., 2003) per se. Broadly, we can divide the belief-based inference methods into two categories: filtering, in which we maintain the belief over the present pose (or a *subset* of the variables of interest); and smoothing, in which we maintain a belief over the entire high-dimensional state, which contains also past poses.

Text-book-standard filters, such as the Extended Kalman Filter (EKF), or the Extended Information Filter (EIF) (Thrun et al., 2004, 2005), are in extensive use for systems in which the dynamics are characterized with additive Gaussian noise. For the more general case, particle filters (Sim et al., 2005) are often the effective choice. These filters maintain lower-dimensional beliefs, in comparison to smoothers, and, by such, their utilization in sequential estimation usually enjoys lower computation times. On the other hand, smoothing approaches enjoy better accuracy and robustness, as maintaining the belief over the entire trajectory allows us to: (1) perform "loop closures", and update the estimate of past poses, when re-observing scenes; (2) perform re-linearization of past constraints, given the updated estimate, and, by such, further improve accuracy; and (3) understand topological properties of the constraint graph. For these reasons, we are encouraged to focus on this estimation paradigm, which is considered state-of-the-art.

The application of this smoothing paradigm to SLAM and robotics was established by Dellaert and Kaess (2006), as "Smoothing and Mapping (SAM)". The authors suggested to represent the belief using a Probabilistic Graphical Model (PGM) – the factor graph, which describes the topology of constraints (factors) (Koller and Friedman, 2009, Dellaert and Kaess, 2017). They then showed that after performing *variable elimination* (Davis, 2006), the belief can be re-factorized and represented with another PGM – the Bayesian Network (Ben-Gal, 2008); this PGM describes the conditional dependencies between the state variables, and from which we can easily derive the MAP estimate. If the factors are linearized, and the noise models are Gaussian (as considered by the EKF), it was shown that this process of finding the MAP estimate is equivalent to solving a linear least square problem; this problem can be solved by calculating the Gaussian belief's upper triangular "square root information matrix" $\boldsymbol{R}$, given through the QR or Cholesky factorizations (Golub and Loan, 1996).

In sequential problems, as time progresses, new constraints, and possibly new variables, are added to the system; then, the belief and state estimate shall be updated. However, propagation of belief over high-dimensional states, has, at worst, cubical complexity (Dellaert and Kaess, 2006), and thus suffers from (what is known as) the "curse of dimensionality" (Bellman et al., 1957). State-of-the-art SAM approaches, such as iSAM2 (Kaess et al., 2012), and SLAM++ (Ila et al., 2017), partially alleviate this concern, by recognizing that that factorized representation of the belief (given as $\boldsymbol{R}$ or as the Bayesian network), can be incrementally and efficiently updated, instead of re-computed "from scratch". In this case, we should only perform partial re-factorization of the belief, starting the elimination from the first variable to receive a new constraint, according to the variable order. Such incremental updates are very efficient when new constraints involve only recent state variables, but can still be challenging when involving variables from the past (i.e., on "loop closures").

1.2.2 Decision Making under Uncertainty

Following the previous discussion, we can now confidently examine the problem of online decision making under uncertainty (Kochenderfer, 2015), which is the actual interest of this work. When solving this problem, we wish to select (online) an action sequence or a policy, out of a set of possible candidates, which we predict to maximize some expected objective when applied on the current state (considering the current belief). Since we consider stochastic (i.e., noisy) actions and observations, this problem is often modeled as (a variation of) the Markov Decision Process (MDP), or Partially Observable MDP (POMDP) (Kaelbling et al., 1998, Thrun et al., 2005). Finding the optimal policy in a PO/MDP is not trivial, as the candidates' rewards are stochastic, and depend on the uncertain outcomes of the actions, and acquired observations. While it might be theoretically possible to approximate the expected objectives by sampling and propagating *state* particles along each predicted trajectory (Thrun, 2000, Stachniss et al., 2005), achieving a sufficient representation of a belief over the high-dimensional state would require an outstanding amount of such particles, making this approach unfeasible. We hence focus here only on parametric belief propagation, as previously discussed. Also, examining the belief space, instead of the state space, which is examined in standard PO/MDPs, allows us to consider belief-based policies, and belief-based information-theoretic reward functions. This paradigm is thus useful for solving tasks such as information gathering, and active-SLAM (Stachniss et al., 2004, Kim and Eustice, 2014), where we wish to reduce the uncertainty over the entire maintained trajectory and/or feature map.

Thus, to evaluate the candidates, and select the optimal one, we shall predictively propagate the belief (usually) multiple time-steps into the future, according to (possibly) different hypotheses for each candidate; in other words, we shall solve multiple "parallel" state inference sessions, while predicting the future belief development, as if we were to follow each candidate. Essentially, this process means performing a forward search in the belief space, and we thus refer to this problem as Belief Space Planning (BSP) (Bonet and Geffner, 2000). Unfortunately, BSP is computationally challenging for online solution, and reducing its cost is of great importance.

1.3 Contribution

The previous discussion leads us to our main goal – allowing computationally efficient decision making in the belief space. Note that in this study, we differentiate between planning and decision making. Planning is a broad concept, which takes into consideration many aspects, such as goal setting and balancing, generation of candidate actions, accounting for different planning horizons and future developments, coordination of agents, and so on. After refining

these aspects, we eventually result in a decision problem: considering an initial state (or belief, in BSP), and a *given* set of candidate actions (or action sequences), we use an objective function to measure the scalar values attained by applying each action on the initial state; to solve the problem, we shall identify the optimal candidate action, which generates the highest objective value. With this rudimentary view-point, we dismiss problem-specific attributes, which allows our formulation to address a wider range of problems.

As explained, the standard solution to the decision problem requires calculation of the objective function for each candidate action. In this work, we suggest to spare this exhaustive calculation, and, instead, identify and solve a *simplified decision problem*, which leads to the same action selection, or one for which the loss in quality of solution can be bounded. A problem may be simplified by adapting each of its components – initial state (belief), objective function, and candidate actions – such that the objective value derivation becomes "easier".

In support of this concept, we present several contributions:

1.3.1 Simplified Decision Making: A Theoretical Framework

We begin by presenting a novel and fundamental theoretical framework, supporting the concept of decision problem simplification. This framework allows us to formally analyze candidate optimality, and quantify the loss induced from possible sub-optimal action selection. Since we address the elementary process of action comparison and selection, the formulation is relevant to general decision problems, and is not tied to decision making in the belief space. The analysis approach relies on utilizing intervals, i.e., upper and lower bounds, over the values of the candidates, to derive loss guarantees; in an extension, we also explain how to utilize confidence intervals, which appear in probabilistic simplifications, to derive probabilistic guarantees. This analysis approach also allow us to perform action elimination. Meaning, instead of exhaustively calculating the exact values of the candidate actions, in order to select the best one, we can begin with their "rough" approximation; our framework would allow us to identify actions which are guaranteed to be sub-optimal, and quickly eliminate them. We could then refine the approximation only for the remaining candidates.

1.3.2 Belief Sparsification

We then show how the idea of problem simplification can be practically applied to BSP problems over high-dimensional states. In this case, we suggest to simplify the problem by considering a sparse approximation of the initial belief, which can be efficiently propagated, in order to calculate the candidates' objective values. Accordingly, we present a scalable belief sparsification algorithm, and analyze its effect on the decision problem using our suggested framework. We even conclude conditions under which the sparsification is guaranteed to induce no loss at all. Furthermore, while several works already utilize belief sparsification to allow long-term operation and tractable state inference, the novelty in our approach is the exploitation of sparsification exclusively and dedicatedly for efficient decision making. After solving the decision problem, the selected action is applied on the *original* belief; by such, we do not compromise the accuracy of the estimated state.

1.3.3 PIVOT

We follow by suggesting another method for reducing the computational complexity of planning in the belief space, by optimizing the process of belief update required for candidate evaluation. As explained, during sequential inference, the order of variables in the belief determines which

variables will be affected by an upcoming update. Potentially, we wish the amount of these affected variables, which are due for re-elimination, to be minimal. Thus, we suggest a simple yet effective idea: before performing the updates during planning, modify the initial belief by performing a precursory standalone variable reordering procedure; this reordering should "push forwards" all loop closing variables, in order to minimize the total number of affected variables by these updates. We call this approach PIVOT: Predictive Incremental Variable Ordering Tactic.

While belief sparsification and PIVOT are essentially related, they are logically different: the first conveys belief *approximation*, while the latter conveys a change of *representation*; this makes PIVOT inherently complementary to various planning methods. Also, while PIVOT is intended to (and does) benefit the efficiency of planning, the tactic often benefits also the efficiency of state inference. If the optimized order is maintain after planning, then the predicted loop closing variables would remain "forwards" in the variable order. Thus, if our prediction coincides with the ground truth during execution, we should similarly reduce the cost of loop closures during inference, when they actually occurs. Further, this tactic is highly relevant when considering sequential re-planning, as the PIVOT order can be applied incrementally before each planning session.

1.3.4 Efficient Modification of the Square Root Matrix

Unfortunately, while new constraints can often be incrementally and efficiently incorporated, changing the order of variables modifies the upper triangular square root matrix $\boldsymbol{R}$ in a non-trivial way, and implies re-factorization of the entire system. The computational cost of such operation is, hence, exceptionally high. Thus, in support of PIVOT, we present an efficient parallelizable algorithm for modification of $\boldsymbol{R}$ on variable reordering. We suggest two algorithmic variants: one, by manipulating the factor directly; and second, with partial (and minimal) re-factorization. Each of the variants holds specific advantages, as we shall compare.

To clarify, with PIVOT we explain how to determine variable order, and with this algorithm we address how to perform the appropriate modification to $\boldsymbol{R}$ practically and efficiently. Nonetheless, we can use this method to apply any desired order. Specifically, even beyond the relevance to real-time operation of autonomous agents, the introduced algorithm is especially valuable for "large-scale" optimization problems (e.g., offline map optimization over extremely large areas), in which full re-factorization is often practically impossible.

1.3.5 Thesis Structure

Each of the contributions is represented with a dedicated chapter. We begin by discussing simplified decision making in Chapter 2 (and extend the discussion to probabilistic simplifications in Appendix A). We follow with a thorough formulation of the BSP problem, in Chapter 3. Chapters 4, 5, and 6 discuss belief sparsification, PIVOT, and square root matrix modification, respectively; experimental demonstration of each of these methods is included at the end of its respective chapter. Appendix B includes proofs to the lemmas an theorems which appear in the main text.

Finally, since the presented belief simplification methods are applicable to general distributions, they can be described through manipulation of probabilistic graphical models (PGMs). Since they are also specifically applicable to the (more limited, but arguably more practical) linearized and Gaussian case, they can also be described through manipulation of matrices. To sacrifice neither generality, nor applicability, the ideas throughout the thesis are presented and explained in both contexts (PGMs and matrices), while pointing to their parallelisms. The readers can follow both explanations, or focus on the variant that is more relevant to them.

1.4 Related Work

Various works explore similar ideas to the ones presented here. In this section we do our best to provide an extensive review of such works, in comparison to ours.

1.4.1 Planning under Uncertainty

First, we want to point out that *planning* is an "umbrella term" for various problems in the fields of AI and robotics. Prominent "planning" problems, which are not the one discussed here, include optimal control and motion planning, in which we search for feasible plans in the configuration space, and often consider perfect knowledge on the world (LaValle, 2006).

When considering planning in the context of PO/MDPs, the research community has been extensively investigating solution methods to provide better scalability for real-world problems. PO/MDP solution approaches are generally divide into "offline" and "online" (Ross et al., 2008). "Offline" methods generally perform global policy optimization, using dynamic programming algorithms, such as, value and policy iteration (e.g., Porta et al., 2006, Pineau et al., 2006). These methods are extremely computationally demanding, especially when considering high-dimensional state space (i.e., search spaces). They are thus not suitable for "online" planning problems, as we consider here, in which we want to infer a specific sequence of actions to be executed immediately.

Instead, when considering "online" scenarios, we typically perform a forward search from the current belief. As implied, standard online POMDP solvers (e.g., Silver and Veness, 2010, Ye et al., 2017) perform search in the state-space, and not the belief space (as we care to do here). Works which do consider planning in the belief space, typically focus on methods for alleviating the search. For example, some solution methods perform direct (localized) trajectory optimization (e.g. Indelman et al., 2015, Van Den Berg et al., 2012). Otherwise, while building on established motions planners (e.g., Karaman and Frazzoli, 2011, Kavraki et al., 1996), works such as the Belief Roadmap (by Prentice and Roy, 2009), FIRM (by Agha-Mohammadi et al., 2014), SLAP (by Agha-mohammadi et al., 2018), and others (e.g., by Patil et al., 2014) rely on sub-sampling a finite graph in the belief space, in which the solution can be searched. The graph created by these methods conveniently preserves the optimal substructure property, which allows to incrementally search for the (approximated) optimal candidate. However, such methods are severely limited, by only allowing propagation of the belief over a single (most-recent) pose through the graph; i.e., they perform low-dimensional pose filtering, rather than high-dimensional belief smoothing, as we do. This forced marginalization of state variables surely compromises the accuracy of the estimation, and limits the applicability to (problems such as) active-SLAM, in which we often wish to examine the information (uncertainty) of the entire posterior state, including the map and/or executed trajectory (Stachniss et al., 2004, Kim and Eustice, 2014).

Nonetheless, as mentioned, we do not focus on generation (or sampling) of candidate trajectories, but, instead, focus on efficient comparison of their objective values, by lowering the cost of belief propagation. Hence, our approach is complementary to the aforementioned graph-based methods, which focus on generating feasible candidates. We demonstrated this compatibility in our experimental evaluation, where we used a graph-based motion planner (from the most recent pose) to simply generate a set of candidate actions; we then efficiently selected the optimal candidate by propagating the simplified (high-dimensional) belief, and evaluating its posterior uncertainty.

1.4.2 Planning with Simplifications

Also, closely related to our approach, several other works examine approximation of the state or the objective function in order to reduce the planning complexity. A recent approach (Bopardikar et al., 2016) suggested using a bound over the maximal eigenvalue of the covariance matrix as a cost function for planning, in an autonomous navigation scenario. Benefits of using this cost function include easy computation, holding an optimal substructure property (incremental search) and the ability to account to misdetection of measurements. Yet, the actual quality of results in terms of final uncertainty, when measured in conventional methods, is unclear. Their usage of bounds in attempt to improve planning efficiency reminds aspects of our work; however, we use bounds to quantify the quality of solution. As they mention in their discussion, an unanswered question is the difference in quality of solution between planning using the exact maximal eigenvalue, and planning using its bound. Our theoretical framework might be able to provide answer to this question.

Boyen and Koller (1998) suggested maintaining an approximation of the belief for efficient state inference. This approximation is done by dividing state variables into a set number of classes, and then using a product of marginals, while treating each class of variables as a single "metavariable". A k-class belief simplification cuts the original exponential inference complexity by a factor of k. The study showed that in rapidly-mixing POMDPs the expectation of the error could be bounded. This simplification method was later examined under a restrictive planning scenario (McAllester and Singh, 1999). The planning was performed using a planning-tree search, in which a constant amount of possible observations was sampled for each tree level, and again assuming a rapidly-mixing POMDP. There, the error induced by planning in the approximated belief space can be bounded as well. This method shares similar objectives with our work, but examines a very specific scenario, which limits its generality.

In the approach described by Roy et al. (2005), the authors attempted to find approximate POMDP solutions by utilizing belief compression, which was done with a PCA-based algorithm. This key idea is very similar to ours, however, in that work, the objective value calculation (i.e., decision making) still relied on the original decompressed belief, instead of the simplified one. Thus, no apparent computational improvement was achieved in planning complexity. The paper also did not make a comparison of this nature, and only presented analysis on the quality of compression.

1.4.3 Belief Sparsification

As mentioned, numerous methods consider sparsification for the probabilistic state inference problem, in order to limit the belief size, and improve its tractability for long-term operation. Although being a well-researched concept, these methods do not examine sparsification in the context of planning problems (influence over action selection, computational benefits, etc.). Thrun et al. (2004), for example, showed that in a SLAM scenario, when using the information filter, forcing a certain sparsity pattern on the belief's information matrix can lead to improved efficiency in belief update. However, they emphasized that the approximation quality was not guaranteed and that certain scenarios could lead to significant divergence.

Also, since Dellaert and Kaess (2006) demonstrated the equivalence between sparse matrices and (factor) graphs for belief representation, graph-based solutions for SLAM problems (which is often a sparse problem) have become more popular. Accordingly, methods for graph sparsification have also gained relevance. For example, Huang et al. (2012) introduced a graph sparsification method, using node marginalization. The resulting graph is notably consistent, meaning, the sparsified representation is not more confident than the original one. Several other approaches (e.g. Carlevaris-Bianco et al., 2014, Carlevaris-Bianco and Eustice, 2014, Kretzschmar

and Stachniss, 2012) suggest to sparsify the graph using the Chow-Liu tree approximation, and show that the KL-divergence from the original graph remains low. Hsiung et al. (2018) reached similar conclusions for fixed-lag Markov blankets. Notably, our sparsification method, which is presented both in matrix and graph forms, preserves the dimensionality of the belief, and only modifies the correlations between the variables. It is also guaranteed to exactly preserve the entropy of the belief.

The approach described by Mu et al. (2017) separated the sparsification into two stages: problem-specific removal of nodes, and problem-agnostic removal of correlations. The authors then demonstrated the superiority of their scheme over agnostic graph optimization, in terms of collision percentage. This two-stage solution reminds the logic in our sparsification method: first, identifying variables with minimal contribution to the decision problem, and then sparsification of corresponding elements. Of course, we use sparsification for planning and not graph optimization.

The work presented by Indelman (2015, 2016) contained the first explicit attempt to use belief sparsification to specifically achieve efficient planning. The papers showed that using a diagonal covariance approximation, a similar action selection could usually be maintained, while significantly reducing the complexity of the objective calculation. This claim, however, is most often not guaranteed. Optimal action selection was only proved under severely simplifying assumptions – when candidate actions and observations only update a single state variable, with a rank-1 update of the information. Nonetheless, this attempt inspired our extensive research and in-depth formal analysis.

1.4.4 Efficient Computation in the Belief Space

The work by Kopitkov and Indelman (2017, 2019) also focused on reducing the computational complexity of planning with high-dimensional beliefs. It showed that when relying on the information gain as an information-theoretic objective for planning, we do not need to explicitly propagate the belief, in order to derive the objective values; instead, we can utilize the matrix determinant lemma, to efficiently update the determinant of the initial belief's covariance matrix. Of course, our method does not limit the objective function used in planning, nor the type of belief distribution (which was, in their case, strictly Gaussian).

Similarly to `PIVOT`, another related work (Farhi and Indelman, 2017) also considers a system-wide view, which jointly examines both the (belief space) planning and the inference processes. There, the authors suggested to re-use calculation from planning during the following state inference sessions, when applying the selected action. We note that while our approach is meant to mainly benefit the planning (and as a "by-product" also the state inference), the intention of that approach is to solely benefit the inference process (as a "by-product" of the planning). Further, while both approaches lead to a reduction in the computational effort required in inference time, their approach requires maintenance of an external cache; in contrast, our method reaches this goal implicitly, by simply modifying the belief representation, and with no explicit intervention in the inference process.

1.4.5 Variable (Re)ordering

One work, by Chaves and Eustice (2016), similarly examined variable ordering in the context of planning, to allow reuse of calculations when examining similar candidate actions or between similar planning sessions. This scheme is conceptually different from the one presented here, as that reordering occurs during the evaluation of the candidates (instead of it being a precursory step), and also requires external Bayes tree caching. Also, while their results are indeed impressive, that work is significantly more restricted than the one we propose here: they consider a

problem-specific scenario (active-SLAM), while we do not; they assume a certain belief structure ("pose graph"), a certain belief distribution (Gaussian), and certain topological properties of the candidate paths, while we make no assumptions of this nature; they also do not account for the possibility of multiple predicted hypotheses per candidate, while we do; and, finally, they only consider planning with predefined control sequences, while we also allow incrementally-evaluated policies. Besides this attempt, we are not aware of other variable ordering approaches designed specifically for planning, as we presented here.

Nevertheless, we recognize that variable ordering tactics (or heuristic) are used extensively in the solution of the state inference problem. Due to the equivalence between graphs and matrices, this concern lies in the intersection of sparse matrix algebra and graph theory (as recognized in SAM and our formulation) (Kepner and Gilbert, 2011). As explained, to solve the inference problem, we need to find the factorized representation of the belief, given either as an "eliminated" Bayesian network, or equivalently, the "square root matrix" $\boldsymbol{R}$ (derived from the QR of Cholseky factorizations). The amount of non-zero entries in $\boldsymbol{R}$, which is known as its "fill-in", reflects the computational cost of this factorization, and relies heavily on the (elimination) order of state variables. Thus, to reduce this cost, we often rely on fill-reducing variable reordering. Although finding the optimal fill-reducing order is NP-complete (Yannakakis, 1981), various heuristics – variations of the minimum degree algorithm, such as COLAMD (Davis et al., 2004), or graph partitioning techniques, such as nested dissection (Gilbert and Tarjan, 1986) – provide empirically good results. Problem-specific heuristics, e.g., for SLAM (Agarwal and Olson, 2012, Krauthausen et al., 2006), which exploit the known state topology, are also applicable.

We recall that in sequential estimation, at each time-step, we shall appropriately update the belief factorization, in order to refine our estimate. To efficiently keep up with such updates, incremental smoothing and mapping methods, such as iSAM2 Kaess et al. (2012) and SLAM++ Ila et al. (2017), apply such fill-reducing orders incrementally, (only) on the subset of re-eliminated variables, during the partial re-factorization. Thus, although the resulting variable order may not be globally optimal, in sequential estimation, variable reordering is usually not applied as a standalone procedure. It should be mentioned that in an earlier version the incremental smoothing and mapping algorithm (iSAM by Kaess et al. (2008)), application of a fill-reducing order was indeed suggested in periodic standalone steps; yet, this approach was later neglected, as it was not cost-efficient in comparison to the incremental application of such orders during updates. Nonetheless, we should clarify that although the suggested PIVOT order is fill-aware, it is not intended to be fill-reducing, but only to benefit the planning process. Also, unlike such ordering tactics, which arrange the variables according to the current belief structure, we uniquely suggest to order the variables based on its predicted future development.

1.4.6 Modification of the Square Root Matrix

We should first clarify that the order of variables in the state should match the order of columns in the square root matrix $\boldsymbol{R}$ (and the "coefficient matrix" $\boldsymbol{J}$, from which it may be derived); hence, reordering of the variables is essentially expressed as column permutation. Numerous methods (e.g., Gill et al., 1974, Davis and Hager, 1996, Chen et al., 2008) examined the modification of $\boldsymbol{R}$ on low-rank updates and downdates, i.e., the addition or removal of rows from/to $\boldsymbol{J}$. However, column permutation is not a low-rank change, making such modification methods inapplicable.

A recent work Touchette et al. (2016) tried to address a similar problem. There, the authors provided symbolic formulas for swapping the order of two consecutive variables. They showed that in such instance, only the corresponding pairs of rows and columns in the square root matrix are affected, while the rest of the matrix entries remain unchanged. They then proposed to use the formulas in a "bubble sort"-like algorithm, to apply permutations involving more variables.

However, this method is only suitable for simple permutations, as for extensive permutations, this "bubble sort" might require up to $O(n^2)$ swaps. The authors even declared that beyond a certain amount of such swaps, one should simply resort back to the belief's information matrix, and re-apply the Cholesky factorization on the permuted matrix. Another drawback of that approach is that every such swap requires to explicitly update all the affected elements. Thus, for extensive permutations, some rows might be completely recalculated over and over. As to be explained, our algorithm is built to avoid this very problem.

As we mentioned, at the arrival of new constraints, the aforementioned incremental inference techniques (Kaess et al., 2012, Ila et al., 2017) incorporate the new constraints by performing partial re-factorization of the belief. During this re-factorization, these methods are able to utilize the update as an opportunity to reorder the re-eliminated variables. We may recall that we also utilize partial re-factorization in one of the two variants of the proposed modification algorithm, similarly to these methods; but, as to be seen, in comparison to them, our algorithm requires re-factorization of a significantly smaller sub-matrix of $\boldsymbol{J}$. Further, these methods are only able to reorder a subset of the variables, starting from the first variable to receive a new constraint. Hence, overall, we are not aware of any approaches which allow to conduct a "standalone" modification of $\boldsymbol{R}$ on global variable reordering, without completely re-factorizing the system.

We note that, like our approach, modern "multi-frontal" implementations of the QR factorization (e.g., Dellaert et al. (2005), Agullo et al. (2013)) also utilize parallelization, by applying multiple transformations (e.g., rotations) at once. However, generally speaking, the parallel fronts are not independent; these fronts dynamically change, and new fronts are initiated according to the results of former transformations. Such multi-frontal implementation requires intricate synchronization. Our algorithm, on the other hand, yields a high-level block division of the rows, such that, by definition, there are no dependencies between the blocks, and there is no need for communication nor synchronization among them; each block of rows can theoretically be processed on a different machine. This approach is also bandwidth efficient, as we do not initiate factorization on the entire matrix, but only process blocks of it at a time. Under this division, each block can be factorized using any implementation of QR, including the aforementioned multi-frontal ones.

1.4.7 (Probabilistic) Action Elimination

Like us, several other works examined action elimination in the context of planning and decision making. In the work by Nakhost and Müller (2010) actions are eliminated from a constructed plan, comprised of a sequence of actions, in order to reduce its execution complexity, but not the planning complexity. Other works (e.g., Rosman and Ramamoorthy, 2012, Sherstov and Stone, 2005, Abel et al., 2015) discuss how to remove actions from consideration, by transferring previously learned knowledge across tasks. Our approach makes no assumption on prior knowledge, and does not even require the decision process to be sequential.

Furthermore, one may associate the probabilistic extension of our theoretical framework with the PAC learning framework in machine learning, which allows formal analysis of the expected generalization error of a learning-from-samples algorithm. This theory is often used in Reinforcement Learning (RL) in the context of the Multi-Armed Bandit (MAB) problem. This is essentially an "exploitation vs. exploration" problem, in which the agent needs to balance learning the reward distribution of each candidate, and exploiting the most promising one. The agent must act (i.e., pull arms) in order to derive reward samples. Various solution algorithms for the problem (e.g., Kakade, 2003, Strehl, 2007, Strehl et al., 2009) define rules for creating such policies. However, we shall emphasize that online planning and RL are two different problems: in MAB, reward samples are generated by *acting* in the environment. On the other hand, we

discuss model-based planning, in which we *predict* each candidates' values, by propagating the belief into the future. Further, sampling is an inherent part of RL and the MAB problem. As the reward models are unknown, we inherently cannot determine the optimal actions with full confidence. We use the PAC framework to analyze the reward-learning progress. In our problem of interest, we know the reward models, thus the solution is theoretically accessible. We rely simplifications to alleviate the expensive calculation of the candidates' values, and use our framework to quantify this approximation error.

Nonethless, we shall specifically mention the works by Even-Dar et al. (2006) and Mannor and Tsitsiklis (2004), which examined action elimination and optimality guarantees in MDPs using Confidence intervals (CIs), in the context of MABs. As mentioned, we also use (confidence) intervals in our analysis; however, we do not utilize this analysis to create sampling strategies, but perform it descriptively, to provide guarantees regarding a returned solution. Further, these works suggest a relatively redundant usage of the CIs: they assume that the intervals are symmetrically positioned around the approximated values, and that the intervals of all candidates are of the same size. This only allows conservative derivation of guarantees and elimination, in comparison to our analysis. Most importantly, we we did not restrict our analysis to sampling-based intervals, as inherently done in RL (and sampling-based POMDP solvers). Our approach intentionally and notably detaches the interval derivation and guarantee formulation. Hence, it can potentially be used to provide guarantees for solvers that rely on other "types" of simplification, rather than sampling, or even when planning with other objectives, such as belief-based or risk-aware rewards.

Chapter 2

Simplified Decision Making: A Theoretical Framework

To begin with, let us consider a *decision problem* $\mathcal{P}$, which we formally define in Definition 1.

Definition 1. A *decision problem* $\mathcal{P}$ is a 3-tuple $(\boldsymbol{\xi}, \mathcal{A}, V)$, where $\boldsymbol{\xi}$ is the *initial state*, from which we examine a *set of candidate actions* $\mathcal{A}$ (finite or infinite), using an *objective function* $V\colon \{\boldsymbol{\xi}\} \times \mathcal{A} \to \mathbb{R}$. Solving the problem means selecting the *optimal action* a^*, such that

$$a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, V(\boldsymbol{\xi}, a). \tag{2.1}$$

According to our suggested solution approach, we wish to generate and solve a simplified yet analogous decision problem $\mathcal{P}_s \doteq (\boldsymbol{\xi}_s, \mathcal{A}_s, V_s)$, which results in the same (or similar) action selection, but for which the solution is more computationally efficient. This can be achieved by altering or approximating any of the problem components – initial state, candidate actions, or objective function – in order to alleviate the calculation of the candidates' objective values. Nonetheless, approximating each of these components represents a different simplification approach. For example, there is a logical difference between simplifying the initial state (i.e., examining different states under the same objective function), and simplifying the objective function (i.e., examining the same state under different objectives); in the first case, we would like to maintain a certain relation between states, and in the second one, a relation between functions.

Next, we will introduce additional ideas to help formalize our goal, and see how these can guide us towards designing effective simplification methods, which are guaranteed to preserve the quality of solution.

2.1 Analyzing Simplifications

2.1.1 Simplification Loss

Examining a simplified decision problem may lead to loss in the quality of solution, when the selected action is not the real optimal action. We can express this loss with the following simplification quality measure:

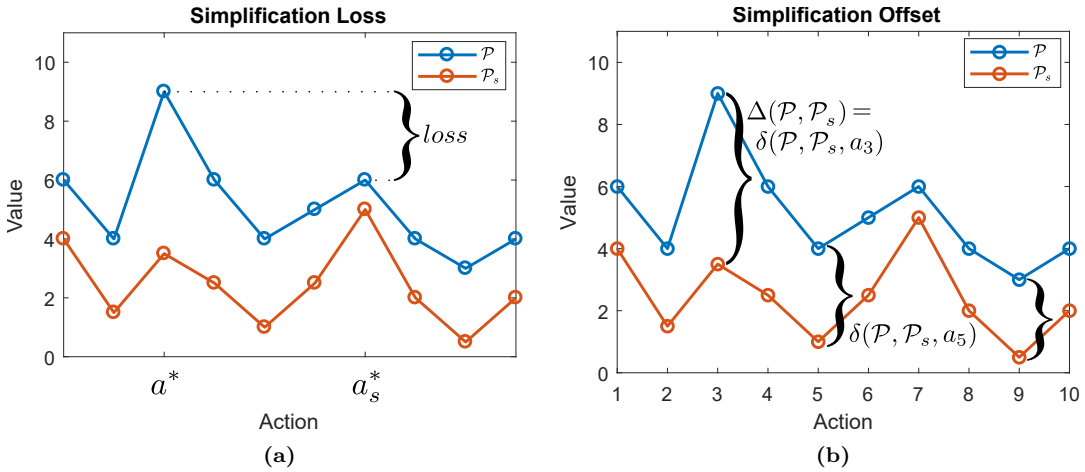


Figure 2.1: $\mathcal{P}_s$ is a simplified version of a decision problem $\mathcal{P}$; the graphs show the objective values of each problem's candidate actions. (a) a_s^* is the optimal action according to the simplified problem, and a^* is the real optimal action; the difference between the (real) objective values of these two actions is the *loss* induced by the simplification. (b) The *offset* measures the maximal difference between *respective* objective values from the two problems, and does not require to explicitly identify a^*/a_s^*.

Definition 2. The *simplification loss* between a decision problem $\mathcal{P} \doteq (\boldsymbol{\xi}, \mathcal{A}, V)$ and its simplified version $\mathcal{P}_s \doteq (\boldsymbol{\xi}_s, \mathcal{A}_s, V_s)$, due to sub-optimal action selection, is

$$loss(\mathcal{P},\mathcal{P}_s) \doteq V(\boldsymbol{\xi},a^*) - V(\boldsymbol{\xi},a_s^*), \quad \text{where } a^* = \underset{a\in\mathcal{A}}{\operatorname{argmax}} V(\boldsymbol{\xi},a),\ a_s^* = \underset{a_s\in\mathcal{A}_s}{\operatorname{argmax}} V_s(\boldsymbol{\xi}_s,a_s). \quad (2.2)$$

To put in words, this loss is the difference between the maximal objective value, attained by applying the optimal candidate action a^* on $\boldsymbol{\xi}$, and the value attained by applying a_s^* (the action returned from the solution of the simplified solution) on $\boldsymbol{\xi}$. This idea is illustrated in Fig. 2.1a. We implicitly assume that the original objective function V can accept actions from the simplified set of candidates $\mathcal{A}_s$. When the solutions to the problems agree $loss(\mathcal{P},\mathcal{P}_s) = 0$.

Most often it is indeed possible to settle for simplified decision problem formulation (which can lead to a sub-optimal action), in order to reduce the complexity of action selection; though, it is important to quantify and bound the potential loss, before applying the selected action, in order to guarantee that this solution can be relied on.

2.1.2 Simplification Offset

To asses the simplification loss, we suggest to identify the *simplification offset*, which acts as an intuitive "distance" measure in the space of decision problems:

Definition 3. The *simplification offset* of a candidate $a \in \mathcal{A}$, between a decision problem $\mathcal{P} \doteq (\boldsymbol{\xi}, \mathcal{A}, V)$, and its simplified version $\mathcal{P}_s \doteq (\boldsymbol{\xi}_s, \mathcal{A}, V_s)$ is

$$\delta(\mathcal{P},\mathcal{P}_s,a) \doteq |V(\boldsymbol{\xi},a) - V_s(\boldsymbol{\xi}_s,a)|. \quad (2.3)$$

Overall, the *simplification offset* between $\mathcal{P}$ and $\mathcal{P}_s$ is

$$\Delta(\mathcal{P},\mathcal{P}_s) \doteq \max_{a\in\mathcal{A}} \{\delta(\mathcal{P},\mathcal{P}_s,a)\}. \quad (2.4)$$

Unlike the loss, the *offset* (which is illustrated in Fig. 2.1b) measures the maximal difference between *respective* objective values from the two problems, and does not require to explicitly identify the optimal actions. This means that the offset, in contrast to the loss, is a property of the simplification *method*, and does not depend on the solution of $\mathcal{P}$ or $\mathcal{P}_s$. It can thus potentially be examined without solving either of the problems, nor calculating V or V_s, as we shall see. Further, from the properties of the absolute value, it is easy to infer that the offset is a valid metric (a distance measure) between decision problems. Intuitively, this means that when the offset between a problem and its simplification is small, then the induced loss should also be small, and the action selection should stay "similar".

For each candidate $a \in \mathcal{A}$, the offset also represents an interval for the real value $V(\boldsymbol{\xi}, a)$, around the respective approximated value $V_s(\boldsymbol{\xi}_s, a)$, in which it must lie, i.e.:

$$V_s(\boldsymbol{\xi}_s, a) - \delta(\mathcal{P}, \mathcal{P}_s, a) \;\leq\; V(\boldsymbol{\xi}, a) \;\leq\; V_s(\boldsymbol{\xi}_s, a) + \delta(\mathcal{P}, \mathcal{P}_s, a) \tag{2.5}$$

Notably, the offset represents only the *size* of this interval, and not its *location* on the value axis (around $V_s(\boldsymbol{\xi}_s, a)$); this is demonstrated in Fig. 2.2a.

Asymmetric Simplification Offset

Surely, when the offset is zero, so is the loss; yet, this statement is not always true the other way around. This practically means that restricting the offset is a more conservative requirement than restricting the loss. To alleviate some of its conservativeness, we can generalize this basic notion, by measuring the lower and upper offsets separately:

Definition 4. The *lower and upper simplification offsets* of a candidate $a \in \mathcal{A}$, between a decision problem $\mathcal{P} \doteq (\boldsymbol{\xi}, \mathcal{A}, V)$, and its simplified version $\mathcal{P}_s \doteq (\boldsymbol{\xi}_s, \mathcal{A}, V_s)$ are

$$\underline{\delta}(\mathcal{P}, \mathcal{P}_s, a) \doteq V_s(\boldsymbol{\xi}_s, a) - V(\boldsymbol{\xi}, a), \tag{2.6}$$

$$\overline{\delta}(\mathcal{P}, \mathcal{P}_s, a) \doteq V(\boldsymbol{\xi}, a) - V_s(\boldsymbol{\xi}_s, a), \tag{2.7}$$

respectively. Overall, the *lower and upper simplification offsets* between $\mathcal{P}$ and $\mathcal{P}_s$ are

$$\underline{\Delta}(\mathcal{P}, \mathcal{P}_s) \doteq \max_{a \in \mathcal{A}} \{\underline{\delta}(\mathcal{P}, \mathcal{P}_s, a)\}, \tag{2.8}$$

$$\overline{\Delta}(\mathcal{P}, \mathcal{P}_s) \doteq \max_{a \in \mathcal{A}} \{\overline{\delta}(\mathcal{P}, \mathcal{P}_s, a)\}, \tag{2.9}$$

respectively.

This extension surely allows us to be less conservative, and more expressive, when describing the effect of simplification methods using the offset; e.g., if the examined simplification is known to always over/underestimate the original objective values, then we know that the upper/lower offset is guaranteed to be zero. Also, this time, for each action $a \in \mathcal{A}$, the combination of the lower and upper offset conveys (the size of) an *asymmetric* interval for the real value $V(\boldsymbol{\xi}, a)$, around the respective approximated value $V_s(\boldsymbol{\xi}_s, a)$, such that

$$V_s(\boldsymbol{\xi}_s, a) - \underline{\delta}(\mathcal{P}, \mathcal{P}_s, a) \;\leq\; V(\boldsymbol{\xi}, a) \;\leq\; V_s(\boldsymbol{\xi}_s, a) + \overline{\delta}(\mathcal{P}, \mathcal{P}_s, a). \tag{2.10}$$

We hence loosely refer to this idea as the "asymmetric offset" (as demonstrated in Fig. 2.2b).

In the following sections, we will demonstrate how we can utilize the offset to infer loss guarantees, through interval overlap analysis (Fig. 2.2c). Note that when defining the offset, we implicitly considered that the two problems examine the same set of candidate actions; this will be valid from now on, unless stated otherwise. Also, for brevity, we will no longer write the initial state as input to V/V_s, nor V, V_s as input to δ/Δ, whenever the context is clear.

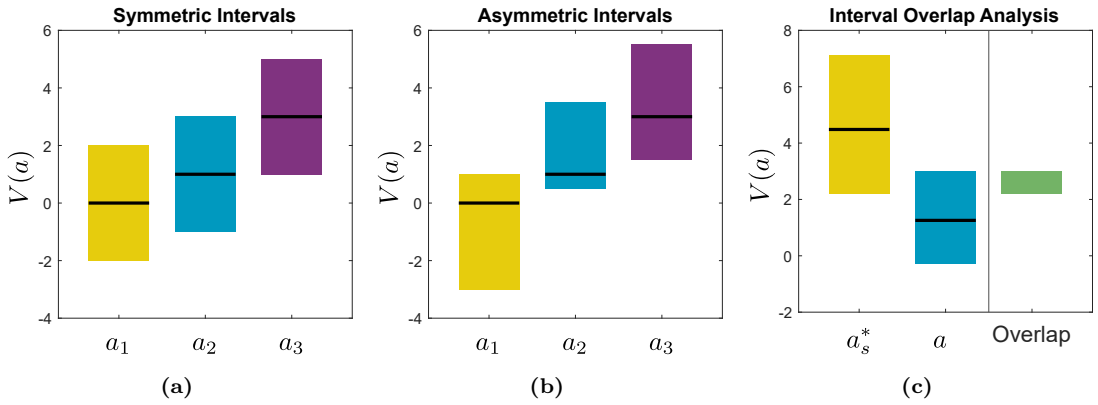


Figure 2.2: The offset of an action a represents the *size* of an interval (colored range) for its value $V(\boldsymbol{\xi}, a)$, around the simplified (approximated) value $V_s(\boldsymbol{\xi}_s, a)$ (black bar); through overlap analysis of such intervals, we are able to derive loss guarantees. (a) The conservative offset Δ represents symmetric and equally-sized intervals for all candidates. (b) By examining the lower/upper offsets $\underline{\delta}/\overline{\delta}$ separately, for each action, we allow the intervals to be asymmetric and of different sizes. (c) This allows us to derive a tightener loss bound, which is determined by the (maximal) overlap between the interval of a^*_s, and the intervals of each of the remaining actions.

Bounding the offset

Obviously, knowing the offset exactly for every action would be equivalent to having access to the original solution. We would thus usually rely on a bound of the offset to infer loss guarantees. As mentioned, the offset measures the difference between respective objective values from the original and simplified problems, and is independent of their solutions. Thus, we can evaluate and attempt to bound the offset *before* solving the problem; by utilizing the general structure of problems in our domain, and knowing how they are affected by the nominative simplification method, we can try to infer a *symbolic formula* for the offset, and draw conclusions from it. This type of analysis often allows us to draw general conclusions regarding the simplification *method*, rather than a specific problem. For example, in Chapter 4, we discuss a novel belief simplification method, used to reduce the cost of planning in the "belief space". By symbolically analyzing the offset (for any decision problem in this domain), we could identify the conditions under which its value is zero, and the simplification is guaranteed to induce no loss. This idea is later demonstrated in Section 4.3.1. Still, we note that providing completely general guarantees, which are valid for all the decision problems in the domain, is not always possible from pure symbolic analysis. Sometimes, to draw decisive conclusions, we must assign the properties of the specific decision problem we wish to solve.

If we failed to reach valuable conclusions from such "pre-solution" symbolic analysis of the offset, we can try to bound it "post-solution", by utilizing the calculated (simplified) values, and (any) known bounds, or limits, for the real objective values; these limits should be selected based on domain knowledge of the specific problem. Then, the following can be easily derived from the definition of the simplification loss:

$$\underline{\delta}(a) \leq V_s(\boldsymbol{\xi}_s, a) - \mathcal{LB}\left\{V(\boldsymbol{\xi}, a)\right\}, \tag{2.11}$$

$$\overline{\delta}(a) \leq \mathcal{UB}\left\{V(\boldsymbol{\xi}, a)\right\} - V_s(\boldsymbol{\xi}_s, a), \tag{2.12}$$

where $\mathcal{LB}$,$\mathcal{UB}$ stand for lower and upper bounds, respectively. We demonstrate how to practically utilize this idea in Section 4.3.2.

2.2 From Offset to Loss Guarantees & Action Elimination

As discussed, our goal is to guarantee that relying on a certain simplification would not induce more than the acceptable loss. As with the offset, bounding the loss can be done on two occasions: (i) **pre-solution evaluation** – this type of analysis occurs *before* solving the simplified problem (based on the availability of "symbolic" offset bounds); and (ii) **post-solution evaluation** – which occurs *after* solving the simplified problem (but before applying the selected action). Surely, we prefer to know if the simplified solution would be worthwhile *before* investing in it; for example, we may consider the case where action execution is costly (as measured with the objective function), and beyond a certain loss, improving the decision making efficiency is not worth the execution of a sub-optimal action. Nonetheless, post-solution guarantees are typically tighter, as we can also rely on the calculated values.

The notion of offset allow us to seamlessly derive both types of guarantees, and easily improve them when refining the solution, or given access to new information.

2.2.1 Post-Solution Loss Guarantees

For a coherent flow, let us first focus on deriving "post-solution" guarantees. Thus, assume we have indeed solved the simplified problem $\mathcal{P}_s$ (i.e., calculated $V_s(a)$, $\forall a \in \mathcal{A}$), and now, before applying the selected action, want to provide guarantees for the loss potentially induced by this solution. As we recall, in our context, the loss is defined as: $V(a^*) - V(a_s^*)$, where a_s^* is the selected candidate. Surely, we do not have access to the real values, but let us consider that for each $a \in \mathcal{A}$, we know its (asymmetric) offset, $\overline{\delta}(a)/\underline{\delta}(a)$, and, by such, the interval in which $V(a)$ lies (as defined in (2.10)). Given this knowledge, we wish to bound the loss of the solution.

By definition, the "highest" interval, i.e., the one located around the maximal $V_s(a)$, corresponds to the selected candidate a_s^* (otherwise, it would not have been selected).

Now, let us assume the real optimal candidate a^* is not actually the selected candidate a_s^*, but another candidate a. From the definition of the offset we know that

$$V(a_s^*) \geq V_s(a_s^*) - \underline{\delta}(a_s^*) \tag{2.13}$$

$$V(a) \leq V_s(a) + \overline{\delta}(a) \tag{2.14}$$

Note that unlike before, these are the upper and lower interval limits of *different* candidates. When the two inequalities hold simultaneously, we can infer that:

$$\mathit{loss}(\mathcal{P}, \mathcal{P}_s) \leq \max\{\epsilon_a, 0\}, \qquad \text{where} \qquad \epsilon_a \doteq \underline{\delta}(a_s^*) + \overline{\delta}(a) - \big(V_s(a_s^*) - V_s(a)\big). \tag{2.15}$$

As we may notice, the loss limit ϵ_a, is determined by the overlap of the interval of a, with the interval of a_s^*. Only if they overlap, i.e.,

$$V_s(a_s^*) - \underline{\delta}(a_s^*) \leq V_s(a) + \overline{\delta}(a), \tag{2.16}$$

then it is possible that the real value of a would be higher than this of a_s^*, and we may experience loss ($\epsilon_a > 0$). Yet, in this case, this loss cannot be greater than the length of the overlapping section. This is demonstrated in Fig. 2.2c. If the pair of intervals does not overlap, i.e., the upper bound for the value of a is lower than the lower bound for the value of a_s^*, i.e.,

$$V_s(a_s^*) - \underline{\delta}(a_s^*) > V_s(a) + \overline{\delta}(a), \tag{2.17}$$

then it implies that ϵ_a should be negative. Thus, a could not actually have been the real optimal action in the first place, and we can infer that this candidate is *guaranteed* to be sub-optimal.

Surely, we do not actually know which candidate is the real optimal candidate a^*. We should thus perform this interval overlap analysis for each candidate $a \neq a_s^*$, assuming it is the optimal candidate. When $\forall a \neq a_s^*$, the pair of intervals do not overlap, it means the real value of a_s^* is higher than the value of any other action. a_s^* would thus be classified as the optimal policy also according to the original solution, and we may derive that the simplified solution optimal. Otherwise, we may return only the most conservative guarantee:

Lemma 1.

$$loss(\mathcal{P}, \mathcal{P}_s) \leq \max\{\epsilon, 0\}, \quad where \quad \boxed{\epsilon \doteq \max_{a \neq a_s^*}\{\epsilon_a\} = \underline{\delta}(a_s^*) - V_s(a_s^*) + \max_{a \neq a_s^*}\left\{\overline{\delta}(a) + V_s(a)\right)\right\}.} \tag{2.18}$$

If we do not have access to a symbolic formula for the offset, and instead rely on the "post-solution offset bounds" (2.11)-(2.12), this expression simplifies to:

$$\epsilon \leq \max_{a \neq a_s^*}\left\{\mathcal{UB}\left\{V(a)\right\}\right\} - \mathcal{LB}\left\{V(a_s^*)\right\}. \tag{2.19}$$

Notably, this analysis allows us to understand not only what is the maximal possible loss, but also which candidates are likely to cause it.

2.2.2 Pre-Solution Loss Guarantees

In the previous analysis we explained how to provide guarantees via post-solution analysis. Now, assume instead that we want to perform pre-solution analysis. Thus, let us now assume that we have access to the offset of each candidate, but *not* to a calculated solution. Looking back at (2.15), we recall that the upper loss limit ϵ_a conveys the overlap of the intervals of a and a_s^*. However, now, we do not have access to the location of the intervals (around the respective $V_s(a)$), as this information is only available after calculating the simplified solution. Hence, we have no choice but be conservative, and assume the intervals are located around the same value, in order to consider the maximal potential overlap for each pair of intervals. Meaning, we can (only) infer that

$$\epsilon_a \leq \underline{\delta}(a_s^*) + \overline{\delta}(a). \tag{2.20}$$

From (2.18), we recall that since we did not know which action is actually a^*, in order to provide the loss guarantee, we had to be conservative, and take the maximal ϵ_a, considering every $a \neq a_s^*$. Now, we also do not know which of the candidates is a_s^*; we thus need to examine the overlap between the intervals of every possible pair of candidates, i.e.,

$$\epsilon_{a',a} = \underline{\delta}(a') + \overline{\delta}(a) \tag{2.21}$$

Then, we can conservatively conclude that

$$loss(\mathcal{P}, \mathcal{P}_s) \leq \max_{a' \neq a}\{\epsilon_{a',a}\} \leq \max_{a \in \mathcal{A}}\{\underline{\delta}(a)\} + \max_{a \in \mathcal{A}}\{\overline{\delta}(a)\} \leq \underline{\Delta} + \overline{\Delta}. \tag{2.22}$$

Indeed, since this conclusion depends only on the definition of $\mathcal{P}_s$, and not on the calculated values, we can reach this conclusion before calculating the solution. We summarize this claim in the Lemma 2:

Lemma 2.

$$\boxed{loss(\mathcal{P}, \mathcal{P}_s) \leq \underline{\Delta} + \overline{\Delta} \leq 2 \cdot \Delta} \tag{2.23}$$

From this discussion we can understand why pre-solution guarantees are always more conservative than guarantees for specific solutions, which can take advantage of additional information (the intervals' locations and identification of the top interval). The offset intuitively allows us to derive guarantees of both kinds.

2.2.3 Action Elimination

We explained how to use the simplification offset to bound the simplification loss. However, sometimes, even after calculating the simplified solution and performing "post-solution" analysis, we still cannot guarantee that the loss is below an acceptable threshold; this is especially true when the available offset bound (i.e., interval size) is not tight enough, or the approximated values (the location of the intervals) are too close to each other. In that case, we would have to refine our solution and/or offset bound, by attempting to utilize a "finer" simplification, or simply resort back to the original problem $\mathcal{P}$ (if no other simplification is available).

Yet, as implied beforehand, in the "post-solution" analysis, we can sometimes identify the candidates which are guaranteed to be sub-optimal, i.e., candidates which cannot possibly be a^*. Thus, before solving the refined (or original) problem, we shall attempt to utilize the current solution to identify candidates which should not be evaluated anymore. In other words, we can use the simplified solution to eliminate, or prune, sub-optimal actions.

As explained in (2.17), a candidate a can be eliminated if its interval does not overlap with the interval of a^*_s. Thus, we can formulate the following lemma:

Lemma 3. *The simplified value of the optimal action a^* must be above the following threshold(s):*

$$\boxed{\tau_1 \leq \tau_2 \leq \tau_3(a^*) \leq V_s(a^*),} \quad \textbf{\textit{where}} \quad \begin{aligned} \tau_1 &\doteq V_s(a^*_s) - 2 \cdot \Delta, \\ \tau_2 &\doteq V_s(a^*_s) - \underline{\Delta} - \overline{\Delta}, \\ \tau_3(a) &\doteq V_s(a^*_s) - \underline{\delta}(a^*_s) - \overline{\delta}(a). \end{aligned} \tag{2.24}$$

An action which does not satisfy this condition is guaranteed to be sub-optimal.

Lemma 3 allows us to characterize the desired optimal candidate a^*, by providing a condition that it must meet. Most importantly, this condition is based solely on the simplified solution and the offset (which, as stated, can often be bounded without having to solve the original problem). The inequality in (2.24) defines three (increasing) thresholds, $\tau_1, \tau_2, \tau_3(a)$ above which $V_s(a^*)$, the simplified value of the real optimal candidate action, must lie; if the simplified value of a candidate appears below any of these thresholds, it can be eliminated. The three thresholds differ by which "variation" of the offset they rely on: τ_1 relies on the conservative symmetric offset, taken over all actions; τ_2 relies on the more expressive upper and lower offsets, taken over all actions; and $\tau_3(a)$ relies on the most expressive per-candidate upper and lower offsets. Surely, the lower thresholds τ_1, τ_2 lead to the most conservative elimination; however, they can possibly be derived more easily, since they do not require per-action evaluation. A visualization of this concept is demonstrated in Fig. 2.3.

Sequential Elimination

This elimination process is especially useful when the simplification method (i.e., initial state and/or value approximation) is scalable. As a rule of thumb, we consider that calculating a rough approximation of the values (large offset) is easier than calculating a refined one (small offset). Hence, given simplifications of different scale (with different offsets from the original problem), one should balance between investing in minimizing the interval size, potentially leading to elimination of more actions, and faster approximation, which might lead to a more lenient elimination.

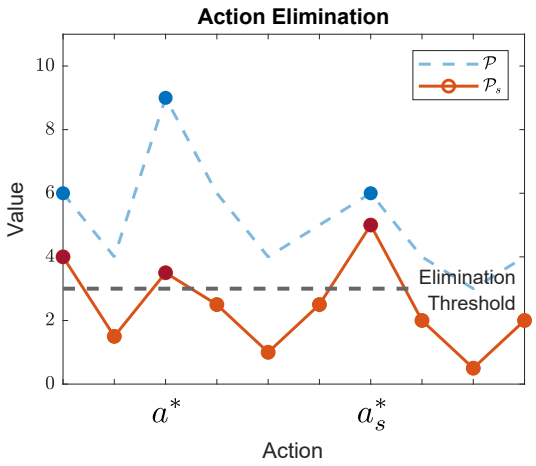


Figure 2.3: Using the solution of the simplified problem, we can eliminate candidate actions that are guaranteed to be sub-optimal. First, we should calculate all the simplified values (in orange), and infer the elimination thresholds (Lemma 3). Only actions for which the simplified value is above these thresholds are viable candidates to be a^*; in this case, only three candidate remain (in red). We can then refine the simplification or calculate the real value only for these non-eliminated candidates (in blue).

Sequentially refining the approximation (decreasing offset) is often the most cost-effective. In that case, we should start with a rough simplification, and perform an elimination iteration. After this iteration, we should try to reduce the size of the interval, using a more refined approximation of the remaining actions' values, and perform another elimination iteration. Each iteration would potentially leave us with a smaller subset of candidates to select from. This way, actions which are clearly sub-optimal are eliminated first, with minimal computational investment, and the refined approximation is only used for the top candidates. Hopefully, after multiple iterations, all candidates but one would be eliminated, as the interval size converges to zero.

If more than one candidate remains, and we cannot eliminate any more actions, we can either solve the original problem considering the remaining actions, or simply return the best choice based on the current approximation. This sequential method is summarized in Algorithm 1. Note that whenever breaking on this anytime algorithm, the loss induced by the selected action is still ensured to be bounded (according to Lemma 2).

Also, we should note that every iteration requires recalculation of the values according to the new scale of approximation. Thus, if the convergence of the elimination interval is very slow, then there is a risk that the overall solution cost might increase, due to the reevaluation of candidates. This situation can be avoided if the approximation method allows to gradually refine these values, instead of recalculating them "from scratch" on every iteration.

2.3 Reducing Simplification Bias

Previously, we suggested the simplification offset as a "distance measure" between decision problems, and recognized that it(s bound) can be used to bound the simplification loss. However, this distance measure may be deceiving, as the problems may appear to be separated by a large offset, even when the simplification induces a small loss. Specifically, this can be the case when the simplification causes a large "bias" in the simplified objective values. In the following section we introduce another concept, to help us handle such scenarios.

Algorithm 1: Sequential action elimination – an "anytime" algorithm.

Inputs:
A decision problem $\mathcal{P} = (\boldsymbol{\xi}, \mathcal{A}, V)$

Output:
A subset of possible candidates $\mathcal{A}'$

1 $\mathcal{A}' \leftarrow \mathcal{A}$
2 select a simplification of the initial state $\boldsymbol{\xi}_s$ and/or the objective function V_s, s.t. $\mathcal{P}_s = (\boldsymbol{\xi}_s, \mathcal{A}, V_s)$
3 calculate or refine $\forall a \in \mathcal{A}'$ its approximated value $V_s(\boldsymbol{\xi}_s, a)$
4 bound or refine the *simplification offset* $\Delta(\mathcal{P}_s, \mathcal{P})$
`// eliminate actions for which the approximated value is below the threshold`
5 **forall** $a \in \mathcal{A}'$ **do**
6 **if** *a does not satisfy inequality (2.24)* **then**
7 $\mathcal{A}' \leftarrow \mathcal{A}'/\{a\}$
8 **if** $|\mathcal{A}'| = 1$ *or* `timeout` **then**
9 **return** $\mathcal{A}'$
10 **else**
11 Refine $\boldsymbol{\xi}_s$ and/or V_s
12 **go to** *3*

2.3.1 Action Consistency

We point out a key observation: to solve the decision problem, we only need to sort (or rank) the candidate actions in terms of their objective function value; changing the values themselves, without changing the order of actions, does not change the action selection. Hence, when two problems maintain the same order of candidate actions, their solution is equivalent. In this case, we can simply say that the two problems are *action consistent*, as demonstrated in Fig. 2.4a.

Definition 5. Two decision problems, $\mathcal{P}_1 \doteq (\boldsymbol{\xi}_1, \mathcal{A}, V_1)$ and $\mathcal{P}_2 \doteq (\boldsymbol{\xi}_2, \mathcal{A}, V_2)$, are *action consistent*, and marked $\mathcal{P}_1 \simeq \mathcal{P}_2$, if the following applies $\forall a_i, a_j \in \mathcal{A}$:

$$V_1(\boldsymbol{\xi}_1, a_i) < V_1(\boldsymbol{\xi}_1, a_j) \iff V_2(\boldsymbol{\xi}_2, a_i) < V_2(\boldsymbol{\xi}_2, a_j). \tag{2.25}$$

If also $V_1 \equiv V_2$, we can simply say that $\boldsymbol{\xi}_1$, $\boldsymbol{\xi}_2$ are *action consistent*, and mark $\boldsymbol{\xi}_1 \simeq \boldsymbol{\xi}_2$.

This relation holds several interesting properties.

Lemma 4. *Action consistency ($\simeq$) is an equivalence relation; i.e., any three decision problems $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, satisfy the following properties:*

1. *Reflexivity:* $\mathcal{P}_1 \simeq \mathcal{P}_1$.
2. *Symmetry:* $\mathcal{P}_1 \simeq \mathcal{P}_2 \iff \mathcal{P}_2 \simeq \mathcal{P}_1$.
3. *Transitivity:* $\mathcal{P}_1 \simeq \mathcal{P}_2 \wedge \mathcal{P}_2 \simeq \mathcal{P}_3 \implies \mathcal{P}_1 \simeq \mathcal{P}_3$.

Lemma 4 implies that the entire space of decision problems is divided into separate equivalence-classes of action consistent problems. Lemma 5 adds that we can transfer between action consistent problems using monotonically increasing functions. We remind again that all proofs are given in Appendix B.

Lemma 5.

$$\mathcal{P}_1 \simeq \mathcal{P}_2 \iff \textit{the mapping } f\colon V_1(\boldsymbol{\xi}_1, a) \mapsto V_2(\boldsymbol{\xi}_2, a) \textit{ is monotonically increasing.} \tag{2.26}$$

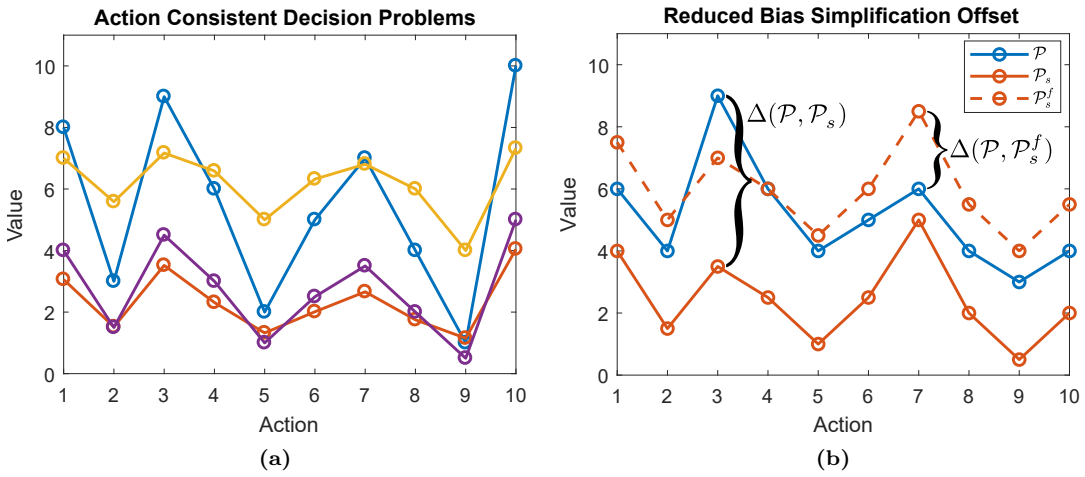


Figure 2.4: (a) Each graph represents the objective values of the candidate actions of a certain decision problem; although the values are different, all the graphs maintain the same trend among the actions, and therefore the problems are action consistent. (b) The simplification offset Δ between $\mathcal{P}$ and $\mathcal{P}_s$ is the maximal difference between the values of *respective* actions. The offset can be reduced by utilizing a monotonically increasing function f (here we used a constant-shift), which leads to an less biased yet action consistent problem $\mathcal{P}_s^f$.

Meaning, if the (scalar) mapping of respective objective values between the two problems agrees with a monotonically increasing function (e.g. a constant shift, a linear transform, or a logarithmic function), then the problems are action consistent. If this mapping is not monotonically increasing, then the problems are not action consistent.

2.3.2 Unbiased Simplification Offset

The notion of action consistency can help us to achieve better guarantees when utilizing our previously developed analysis approach. We now understand that when deriving loss bounds, instead of examining a simplified problem $\mathcal{P}_s$, we can, equivalently, examine any other problem $\mathcal{P}_s^f$ that is action consistent with it. Further, such a problem will necessarily be of the form $\mathcal{P}_s^f \doteq (\boldsymbol{\xi}_s, \mathcal{A}, f \circ V_s)$, where f is monotonically increasing.

Accordingly, instead of examining the simplification offset, as considered thus far, we can examine the *unbiased offset*:

Definition 6. The *unbiased simplification offset* between a decision problem $\mathcal{P} \doteq (\boldsymbol{\xi}, \mathcal{A}, V)$, and its simplified version $\mathcal{P}_s \doteq (\boldsymbol{\xi}_s, \mathcal{A}, V_s)$ is

$$\Delta^*(\mathcal{P},\mathcal{P}_s) \doteq \min\left\{\Delta(\mathcal{P},\mathcal{P}_s^f) \mid f\colon \mathbb{R}\to\mathbb{R} \text{ is monotonically increasing } \wedge\ \mathcal{P}_s^f \doteq (\boldsymbol{\xi}_s, \mathcal{A}, f\circ V_s)\right\}. \tag{2.27}$$

The unbiased offset is the minimal offset between $\mathcal{P}$ and any problem action consistent with $\mathcal{P}_s$. A demonstrative example appears in Fig. 2.4b. Specifically, $\mathcal{P} \simeq \mathcal{P}_s$, if and only if the balanced offset is zero:

Lemma 6.

$$\mathcal{P} \simeq \mathcal{P}_s \iff \gamma^*(\mathcal{P},\mathcal{P}_s) = 0. \tag{2.28}$$

Thankfully, our previous conclusions still hold, and we can use the unbiased simplification offset to bound the loss:

Lemma 7.

$$0 \leq loss(\mathcal{P}, \mathcal{P}_s) \leq 2 \cdot \Delta^*(\mathcal{P}, \mathcal{P}_s). \tag{2.29}$$

Since $\Delta^*(\mathcal{P}, \mathcal{P}_s) \leq \Delta(\mathcal{P}, \mathcal{P}_s^f)$, for *any* monotonically increasing f. We can symbolically develop $\Delta(\mathcal{P}, \mathcal{P}_s^f)$, for *any* such f that is convenient, in order to bound the loss; such a function should help "counter" the effect of the simplification on the objective values. We may also recognize that the unbiased offset satisfies the triangle inequality (like the standard offset):

Lemma 8. *The balanced simplification offset satisfies the triangle inequality:*

$$\Delta^*(\mathcal{P}_1, \mathcal{P}_2) + \Delta^*(\mathcal{P}_2, \mathcal{P}_3) \geq \Delta^*(\mathcal{P}_1, \mathcal{P}_3). \tag{2.30}$$

This property can potentially help in bounding the loss, when applying multiple simplifications. However, unlike the standard offset, the unbiased offset is scaled according to the original objective values (like the loss), and is asymmetric in its input arguments. It is, therefore, not considered a metric; still, the aforementioned properties, along with the obvious non-negativity, make the unbiased offset a quasi-metric (or asymmetric metric), which induces an appropriate topology on the space of decision problems, as explained by Künzi (2001).

Finally, in Appendix A we further extend our theoretical framework to allow derivation of probabilistic guarantees, (e.g.) when facing probabilistic simplifications.

Related concepts We may also note that the notions of action consistency and simplification offset are related to the concept of "rank correlation" – a scalar statistic which measures the correlation between two ranking vectors (see Kendall, 1948). Yet, such ordinal vectors are oblivious to the cardinal objective values, and, therefore, cannot be used to bound the simplification loss. The rank correlation coefficient mostly serves for statistical analysis, as its calculation requires perfect knowledge on the ranking vectors. Since the rank variables are not independent of each other, a change or addition of a single vector entry may subsequently lead to change in all other entries, and require complete recalculation of the correlation coefficient. On the other hand, the concepts we introduced rely on a "local relation" between the problems: to check for action consistency, we only examine pairs of actions at a time; and to evaluate the offset – only pairs of respective objective values. Addition of candidates, for example, does not affect these relations between the existing candidates. As we explain next, this locality can be utilized to derive offset and loss bounds. It is also worth mentioning that the idea of examining only the order of candidate actions, instead of their cardinal objective values, sometimes appears in the context of economics under the term *ordinal utility* (e.g., Manski, 1988); this term, however, is not prominent in the context of artificial intelligence.

Chapter 3

Decision Making in the Belief Space: Problem Definition

3.1 Belief Maintenance: Graphical View

3.1.1 Probabilistic State Inference

We consider a sequential stochastic optimization process. In such a process, we wish to sequentially refine an estimate of a state vector of random variables, given a stream of stochastic constraints over these variables ("factors").

For example, in our context of mobile robots, we may consider the following system: at time-step k, an agent at pose x_{k-1} transitions to pose x_k, using a control u_k, and then collects an observation z_k. The agent's state $\mathcal{X}_k \doteq \{x_0, \ldots, x_k\} \cup \mathcal{L}_k$ is comprised of its entire trajectory up to that point, alongside (optionally) a collection of additional external variables $\mathcal{L}_k$ (such as the positions of observed landmarks). We assume that the pose transition and the observations comply to the following models, respectively:

$$x_k = g_k(x_{k-1}, u_k) + w_k, \tag{3.1}$$
$$z_k = h_k(\mathcal{X}_k^o) + v_k, \tag{3.2}$$

where $\mathcal{X}_k^o$ represents the subset of the state variables which were observed ("involved in the observation"); g_k and h_k are deterministic functions; and w_k and v_k are independent random variables, representing stochastic noise. We may also assume access to prior probabilistic knowledge on the initial pose, given as

$$x_0 = \overline{x_0} + \boldsymbol{v}_0, \tag{3.3}$$

where $\overline{x_0}$ is an initial estimate, and v_0 represents stochastic noise. Given these models, state transitions and observations (and the prior) induce a set of probabilistic constraints among the variables, which we refer to as *factors*:

$$f_{u_k}^{\text{motion}}(x_{k-1}, x_k) \doteq \mathbb{P}(x_k \mid x_{x-1}, u_k), \tag{3.4}$$
$$f_{z_k}^{\text{obs}}(\mathcal{X}_k^o) \doteq \mathbb{P}(z_k \mid \mathcal{X}_k^o), \tag{3.5}$$
$$f_{\overline{x_0}}^{\text{prior}}(x_0) \doteq \mathbb{P}(x_0). \tag{3.6}$$

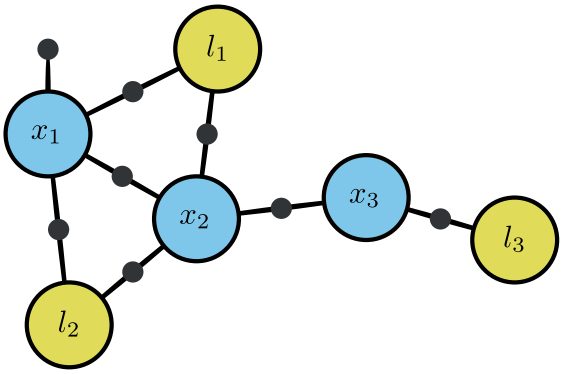


Figure 3.1: An exemplary factor graph representing the belief of a SLAM agent. The state contains two types of variables: poses (blue nodes) and landmarks (yellow nodes). Motion and observation factors (small black nodes) connect the variables.

At each time-step, the agent would like to maintain the posterior distribution over its state, given the controls and observations taken until that time:

$$b_k(\mathcal{X}_k) \doteq \mathbb{P}(\mathcal{X}_k \mid \mathcal{U}_k, \mathcal{Z}_k), \tag{3.7}$$

where $\mathcal{U}_k \doteq \{u_1, \dots, u_k\}$ and $\mathcal{Z}_k \doteq \{z_1, \dots, z_k\}$. This high-dimensional joint distribution is known as the agent's *belief*. The estimation of this belief and its parameters is known as probabilistic state inference. We can utilize Bayes' rule and Markov assumption ("Bayesian inference"), to represent the belief as the product of the prior, and the observation and pose likelihoods:

$$b_k(\mathcal{X}_k) \propto \mathbb{P}(x_0) \cdot \prod_{i=1}^{k} \mathbb{P}(x_i \mid x_{i-1}, u_i) \cdot \prod_{i=1}^{k} \mathbb{P}(z_i \mid \mathcal{X}_k^o). \tag{3.8}$$

Conveniently, this expression is exactly a multiplication of the factors defined by the aforementioned models:

$$b_k(\mathcal{X}_k) \propto f_{\overline{x_0}}^{\text{prior}}(x_0) \cdot \prod_{i=1}^{k} f_{u_i}^{\text{motion}}(x_{i-1}, x_i) \prod_{i=1}^{k} f_{z_i}^{\text{obs}}(\mathcal{X}_k^o) \tag{3.9}$$

By looking at (3.9), instead of (3.8), we no longer explicitly consider $\mathcal{U}_k$ or $\mathcal{Z}_k$ (whose values are known); these are used implicitly to define the factors, which allow us to focus only on estimating the (unknown) variables of interest. Further, this factorization of the belief can be represented with a common Probabilistic Graphical Model (PGM) – the factor graph $FG_k \doteq (\mathcal{X}_k \cup \mathcal{F}_k, \mathcal{E}_k)$, where the set of nodes is comprised of the set of state variables $\mathcal{X}_k$, and the set of factors $\mathcal{F}_k$; and the undirected edges in $\mathcal{E}_k$ are used to connect state variables to the factors in which they are involved. An exemplary factor graph is given in Fig. 3.1.

We are typically interested in evaluating the probabilities of different state estimates, and, specifically, finding the maximum-a-posteriori (MAP) estimate $\mathcal{X}_k^* \doteq \operatorname{argmax}_{\mathcal{X}_k} b_k(\mathcal{X}_k)$ ("MAP inference"). To do so, we can *re-factorize* the belief $b_k(\mathcal{X}_k)$ (from (3.9)) to a product of $|\mathcal{X}_k|$ conditional factors, representing the conditional probabilities of each state variable, given (a subset of) the other variables. This process is known as *variable elimination*.

While the order of state variables used in (3.9) (and the factor graph) is arbitrary, the variable order now plays an important role, as it conveys the elimination order chosen for the

variables. By setting a certain order, we can arrange the state variables into a state vector $\boldsymbol{X}_k \doteq [\boldsymbol{X}_k(1), \ldots, \boldsymbol{X}_k(n_k)]^T$, where $n_k \doteq |\mathcal{X}_k|$ is the state size. Then, in short, this elimination process is done as follows: we choose the first variable for elimination $\boldsymbol{X}_k(1)$, and select all the factors in $\mathcal{F}_k$, in which it is involved. Then, we shall utilize the chain rule to replace the selected factors with two "new" factors: (a) the conditional probability of $\boldsymbol{X}_k(1)$, given all the other variables involved in the selected factors $\mathbb{P}(\boldsymbol{X}_k(1) \mid d(\boldsymbol{X}_k(1)))$; and (b) a "marginal factor" $f_{k,1}^{\text{marginal}} \doteq \mathbb{P}(d(\boldsymbol{X}_k(i)))$, which connects all these variables (not including $\boldsymbol{X}_k(1)$). We then repeat this process according to the elimination order, examining at each step the remaining factors from $\mathcal{F}_k$, and the accumulated marginal factors. Finally, we shall be left with a product of n_k conditional factors:

$$b_k(\boldsymbol{X}_k) \propto \prod_{i=1}^{n_k-1} \mathbb{P}(\boldsymbol{X}_k(i) \mid d(\boldsymbol{X}_k(i))) \cdot \mathbb{P}(\boldsymbol{X}_k(n_k)), \tag{3.10}$$

where $d(x)$ denotes the set of variables x depends on (for the chosen variable order). Surely, a variable may only depend on those which follow it according to the chosen order. Then, from this hierarchical representation, we can derive the MAP estimate (or other marginal distributions), by examining the variables in the reverse order.

Graphically, this factorization of the belief to conditional probabilities can be represented with yet another PGM – a Bayesian network ("Bayes net") $BN_k \doteq (\mathcal{X}_k, \mathcal{D}_k)$. In the Bayes net, the directed edges ($\mathcal{D}_k$) represent the conditional dependencies between the connected variables, such that the edge's destination depends on its origin. The elimination process can be seen as building the belief's Bayes net from its factor graph. The Bayes net, and the elimination process are demonstrated in Fig. 3.2.

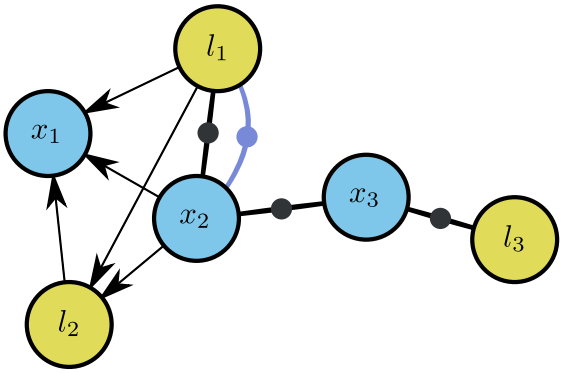
Choosing different orders for the variables has a significant influence on the resulting factorization. As the Bayes net's edges represent also indirect dependencies among the variables, it is typically denser (i.e., has more edges) than the factor graph. Edges between variables in the Bayes net, which did not have a connecting factor in the Factor graph, are known as "fill-in". The amount of fill-in reflects the computational cost of elimination, and different elimination orders may result in vastly different fill-in, as we later review.

To clarify, any other factorization of the belief to a product of conditional probabilities, and, specifically, the initial factorization given in (3.8), can also be represented with a Bayes net. However, as mentioned, this network would: (1) also include nodes for the known variables ($\mathcal{U}_k$ and $\mathcal{Z}_k$); (2) represent more than n_k factors; and (3) not necessarily convey a unique order of dependency. Such network does not serve us in the following discussion; hence, from now on, whenever we mention "the belief's Bayes net", we will refer to the one matching the eliminated representation given in (3.10).

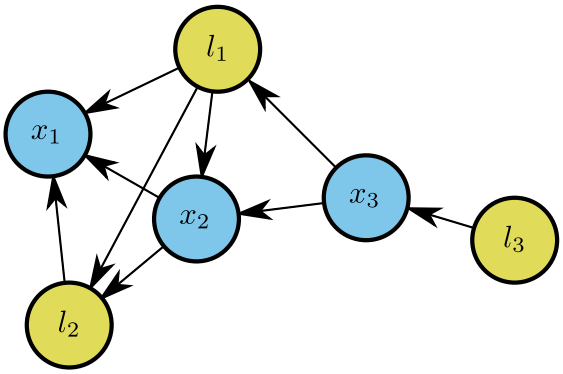
3.1.2 Incremental State Inference

We recall that we consider a sequential problem, in which the system (i.e., sets of variables and factors) may grow as time progresses. Hence, at the arrival of new factors, we would wish to update the belief accordingly, and refine our estimate.

Formally, consider the initial belief $b_k(\mathcal{X}_k)$, represented with FG_k and BN_k, and assume that at time-step $k+1$ we wish to update the system with new factors, and possibly new state variables (e.g., transition to a new pose). Such an update can be represented as a factor (sub-)graph, which we wish to merge into the initial graph; i.e., $FG_{k+1}^{\delta} \doteq (\mathcal{X}_k \cup \mathcal{X}_{k+1}^{\delta} \cup \mathcal{F}_{k+1}^{\delta}, \mathcal{E}_{k+1}^{\delta})$, such that $FG_{k+1} \doteq FG_k \cup FG_{k+1}^{\delta}$. Note that the set of nodes in FG_{k+1}^{δ} also contains the state variables from the previous time-step k, as new factors can, of course, involve variables from



(a) After eliminating variables x_1 and l_2. New "marginal" factors (in purple) might be added to the graph in intermediate elimination steps.



(b) After elimination of all the variables.

Figure 3.2: Creating a Bayes net from the factor graph given in Fig. 3.1, by performing variable elimination; this process depends on the elimination order, which here is chosen to be $[x_1, l_2, x_2, l_1, x_3, l_3]$. Arrows between the variable nodes represent conditional dependencies.

the past (otherwise, the initial graph and the update would be disconnected). As mentioned, re-observing variables from the (non-immediate) past is commonly known as "loop closing".

Adding new factors to the belief (i.e., updating (3.9) or the matching factor graph) is trivial when utilizing Bayes' rule:

$$b_{k+1}(\mathcal{X}_{k+1}) \doteq \mathbb{P}(\mathcal{X}_{k+1} \mid \mathcal{U}_{k+1}, \mathcal{Z}_{k+1}) \propto b_k(\mathcal{X}_k) \cdot \mathbb{P}(x_{k+1} \mid x_k, u_k) \cdot \mathbb{P}(z_{k+1} \mid \mathcal{X}^o_{k+1}); \tag{3.11}$$

yet, inferring the updated factorized belief representation at time $k+1$ (i.e., updating (3.10) or the matching Bayes net) is not, and requires re-elimination. Conveniently though, this can be performed incrementally, given access to the elimination process of the belief at the previous time-step k.

By looking at FG^{δ}_{k+1}, we can identify the set of involved variables marked as $\mathcal{I}nv(FG^{\delta}_{k+1})$: these are the state variables which are directly involved in any of the factors $\mathcal{F}^{\delta}_{k+1}$. By definition, all the newly added variables $\mathcal{X}^{\delta}_{k+1}$ are involved, and also a certain subset of the "former" state variables $\mathcal{X}_k$. Now, assume that $X_k(j)$ is the first variable in $\mathcal{X}_k$, according to the elimination order $\boldsymbol{X}_k$, that is involved in any of the new factors (i.e., $\in \mathcal{I}nv(FG^{\delta}_{k+1})$).

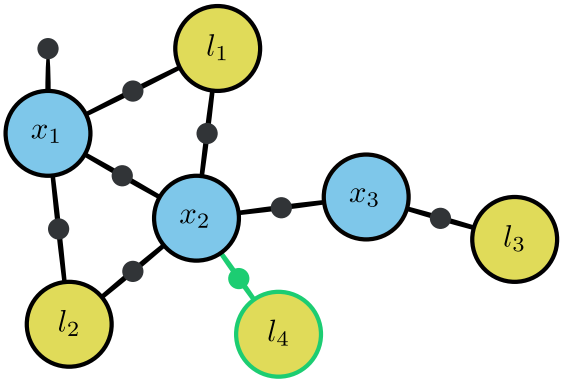
According to the elimination algorithm explained before, we can conclude that starting the elimination process with variables $\boldsymbol{X}_k(1)$ to $\boldsymbol{X}_k(j-1)$ would be equivalent in both FG_k and FG_{k+1}, and result in the same conditionals (and marginals factors); the addition of the new factors would only affect the elimination of the remaining variables. So, instead of performing full elimination of b_{k+1}/FG_{k+1}, we can potentially reuse the conditionals (and marginals factors) calculated in the elimination process at time-step k, and incrementally update the factorization. In other words, we can (re)start the elimination process from the elimination of the first involved variable:

$$b_{k+1}(\mathcal{X}_k) \propto \prod_{i=1}^{j-1} \mathbb{P}(\boldsymbol{X}_k(i) \mid d(\boldsymbol{X}_k(i))) \cdot \prod_{f \in \mathcal{F}^{\text{marginal}}_{k,j-1}} f \cdot \prod_{f \in \mathcal{F}^{\text{remain}}_{k,j-1}} f \cdot \prod_{f \in \mathcal{F}^{\delta}_{k+1}} f, \tag{3.12}$$

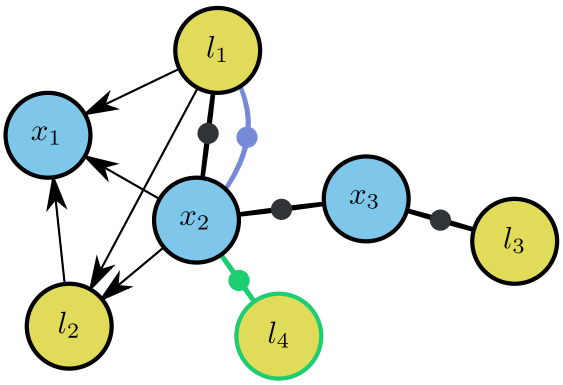
where the first product contain the first $j-1$ conditionals calculated at time-step k; $\mathcal{F}^{\text{marginal}}_{k,j}$ are the (remaining) marginal factors accumulated after eliminating the first $j-1$ variables (at time-step k); $\mathcal{F}^{\text{remain}}_{k,j-1}$ are the subset of factors $\mathcal{F}_k$, which involve only variables $\boldsymbol{X}_k(j)$ to $\boldsymbol{X}_k(n_k)$, and are left for elimination after eliminating the first $j-1$ variables (at time-step k); and $\mathcal{F}^{\delta}_{k+1}$ are the added factors at time-step $k+1$. A graphical demonstration of this concept is provided in Fig. 3.3. Of course, such incremental calculation requires caching of the marginal factors at each step.

We refer to the variables which are left for re-elimination, i.e., $\boldsymbol{X}_{k+1}(j)$ to $\boldsymbol{X}_{k+1}(n_{k+1})$, as the "affected variables" (by this update). Of course, to take advantage of such incremental updates, the (elimination) order of the first $j-1$ variables in $\boldsymbol{X}_{k+1}$ (the "unaffected variables") must match the one set in $\boldsymbol{X}_k$ – otherwise, they would become affected too. Nonetheless, as we shall later discuss, we may, and often do, "shuffle" the order of the affected variables (which, in any case, include the new variables) during this re-elimination. It is hence sensible that the order of variables would dynamically change over time.

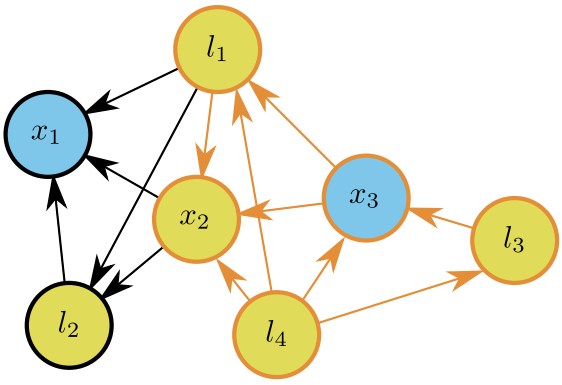
We should also mention that by using more advanced graphical structures, such as the Bayes or Junction tree (Kaess et al., 2010), it is sometimes possible to avoid re-elimination of some of the variables we defined as "affected" (i.e., all the variables following the first involved). Since this aspect is not essential to the discussion, we choose not to over-complicate this formulation, and assume all affected variables are due to re-elimination.



(a) The factor graph from Fig. 3.1, after addition of a new landmark, and a new factor (in green).



(b) We can "restart" the original elimination process from the elimination of the first involved variable (here, x_2), and add the new factors.



(c) The only variables affected by the update (in orange) are the first involved variable, and those which follow it, according to the elimination order.

Figure 3.3: Incrementally updating the Bayes net (i.e., belief factorization) after addition of new factors and/or variables to the factor graph.

3.2 Belief Maintenance: Matrix View

3.2.1 Square-Root SAM

In the previous discussion we made no assumptions on the type of distributions of the probabilistic factors. Nonetheless, a particularly interesting case, which we will focus on in this paper, is when assuming Gaussian prior and model noise, such that

$$w_k \sim \mathcal{N}(0, \boldsymbol{W}_k), \tag{3.13}$$
$$v_k \sim \mathcal{N}(0, \boldsymbol{V}_k), \tag{3.14}$$
$$v_0 \sim \mathcal{N}(0, \boldsymbol{V}_0), \tag{3.15}$$

where $\forall k \in \mathbb{N}$, $\boldsymbol{W}_k, \boldsymbol{V}_k$ are the covariance matrices of the respective zero-mean Gaussian noise variables $\boldsymbol{w}_k, \boldsymbol{v}_k$, and $\boldsymbol{V}_0$ is the covariance matrix of the prior over the pose x_0.

Assigning Gaussian probability density functions in (3.8) yields

$$b_k(\mathcal{X}_k) \propto \exp\left[-\frac{1}{2}\|x_0 - \overline{x_0}\|^2_{\boldsymbol{V}_0}\right] \cdot \prod_{i=1}^{k} \exp\left[-\frac{1}{2}\|x_i - g_k(x_{i-1}, u_i)\|^2_{\boldsymbol{W}_i}\right] \cdot \prod_{i=1}^{k} \exp\left[-\frac{1}{2}\|z_k - h_k(\mathcal{X}^o_k)\|^2_{\boldsymbol{V}_i}\right], \tag{3.16}$$

where $\|\square\|_\Sigma \doteq \sqrt{\square^T \Sigma^{-1} \square}$ marks the Mahalanobis distance. Then, to find the MAP estimate of this belief at time-step k, we shall solve:

$$\mathcal{X}^*_k = \underset{\mathcal{X}_k}{\operatorname{argmin}} \left\{ \|x_0 - \overline{x_0}\|^2_{\boldsymbol{V}_0} + \sum_{i=1}^{k} \|x_i - g_k(x_{i-1}, u_i)\|^2_{\boldsymbol{W}_i} + \sum_{i=1}^{k} \|z_k - h_k(\mathcal{X}^o_k)\|^2_{\boldsymbol{V}_i} \right\}. \tag{3.17}$$

This is a (generally non-linear) least squares problem. To solve it, if the factors are not linear, we can utilize iterative local optimization. We begin by linearizing the non-linear functions, by calculating their Jacobian matrices, around an initial state estimate $\overline{\boldsymbol{X}_k}$ (typically the MAP estimate from time-step $k-1$, with proper initialization for new variables). Note that to calculate these Jacobians, the order of variables, which matches the order of columns in the matrices, can no longer be arbitrary; we again use $\boldsymbol{X}_k$ to mark the state vector, which represents the order of state variables. Given these linearized models, we can easily derive that the belief at time-step k (from (3.16)), is also (approximately) Gaussian:

$$b_k(\boldsymbol{X}_k) \approx \mathcal{N}(\boldsymbol{X}^*_k, \boldsymbol{\Lambda}^{-1}_k), \tag{3.18}$$

where $\boldsymbol{\Lambda}_k$ is the information/precision matrix, achieved by calculating $\boldsymbol{\Lambda}_k \doteq \boldsymbol{J}^T_k \boldsymbol{J}_k$, where

$$\boldsymbol{J}_{k\geq 1} \doteq \begin{bmatrix} \boldsymbol{J}_0 \\ \boldsymbol{J}^\delta_1 \\ \vdots \\ \boldsymbol{J}^\delta_k \end{bmatrix}, \boldsymbol{J}_0 \doteq \left[\boldsymbol{V}_0^{-\frac{1}{2}} \boldsymbol{P}_0\right], \boldsymbol{J}^\delta_{k\geq 1} \doteq \begin{bmatrix} \boldsymbol{W}_k^{-\frac{1}{2}} \boldsymbol{G}_k \\ \boldsymbol{V}_k^{-\frac{1}{2}} \boldsymbol{H}_k \end{bmatrix}, \tag{3.19}$$

$g'_k(x_{k-1}, u_k, x_k) \doteq -x_k + g_k(x_{k-1}, u_k)$, $\boldsymbol{G}_k \doteq \nabla|_{\overline{\boldsymbol{X}}_k} g'_k$, $\boldsymbol{H}_k \doteq \nabla|_{\overline{\boldsymbol{X}}_k} h_k$, $\boldsymbol{P}_0 \doteq \nabla|_{\overline{\boldsymbol{X}}_k} p_0$. and $\square^{-\frac{1}{2}}$ marks the Cholesky factor of the matrix.

Also, it can then be shown (as thoroughly examined by Dellaert and Kaess (2006)) that, in this case, the MAP estimate is given as $\boldsymbol{X}_k^* = \overline{\boldsymbol{X}_k} + \boldsymbol{\eta}_k^*$, where $\boldsymbol{\eta}_k^*$ – the "correction" to the initial estimate – is achieved by solving the following system of linear equations:

$$\boldsymbol{J}_k \cdot \boldsymbol{\eta}_k = \boldsymbol{\zeta}_k, \tag{3.20}$$

with the Right Hand Side (RHS) vector $\boldsymbol{\zeta}_k$, given as

$$\boldsymbol{\zeta}_{k\geq 1} \doteq \begin{bmatrix} \boldsymbol{\zeta}_0 \\ \boldsymbol{\zeta}_1^\delta \\ \vdots \\ \boldsymbol{\zeta}_k^\delta \end{bmatrix}, \; \boldsymbol{\zeta}_0 \doteq \left[\overline{x_0}\right], \; \boldsymbol{\zeta}_{k\geq 1}^\delta \doteq \begin{bmatrix} \overline{x_k} - g(\overline{x_{k-1}}, u_k) \\ z_k - h(\overline{\mathcal{X}_k^o}) \end{bmatrix}. \tag{3.21}$$

After finding the solution, we may repeat the process by re-linearizing (some of) the factors around the updated estimate, and re-solving the linear system, until sufficient convergence.

We note that since every linear constraint (i.e., row in the Jacobian matrix $\boldsymbol{J}_k$) represents a factor from $\mathcal{F}_k$, and the order of columns in the matrix matches the variable order in $\boldsymbol{X}_k$, the sparsity pattern of $\boldsymbol{J}_k$ matches the structure of the factor graph FG_k: looking at each row of the matrix, entries in it are Non-Zero (NZ), if and only if the variables matching the column-index of these entries are involved in the corresponding factor. Further, the solution of this system is achieved by calculating the upper triangular square root information matrix $\boldsymbol{R}_k$, and then performing "backwards/forwards substitution" (see Golub and Loan, 1996). This matrix is given by the Cholesky factorization (marked `chol`) of $\boldsymbol{\Lambda}_k$, such that $\boldsymbol{\Lambda} \doteq {\boldsymbol{R}_k}^T \boldsymbol{R}_k$, or, equivalently, the QR factorization (marked `qr`) of $\boldsymbol{J}_k$, such that $\boldsymbol{J}_k = \boldsymbol{Q}_k \boldsymbol{R}_k$ for an orthogonal matrix $\boldsymbol{Q}_k$; these factorizations are visualized in Fig. 3.4.

When considering linear(ized) Gaussian models, there is a duality between the factorization of $b_k(\boldsymbol{X}_k)$ from a product of $|\mathcal{F}_k|$ factors, to a product of n_k conditional probabilities, and the factorization of $\boldsymbol{J}_k$ ($|\mathcal{F}_k|$ rows) to $\boldsymbol{R}_k$ (n_k rows). Each conditional probability corresponds to the respective row of $\boldsymbol{R}_k$, such that the sparsity pattern of this matrix represents the Bayes net BN_k: the variable at index i is dependent on the variable at index j, if and only if $\boldsymbol{R}_{k\,ij}$ is NZ.

Accordingly, NZ entries in the square root matrix $\boldsymbol{R}_k$, which were zero entries in the information matrix $\boldsymbol{\Lambda}_k$, are (also) known as "fill-in". The amount of fill-in is directly related to the computational cost of the factorization, and hence we care to minimize it. As previously implied, this fill-in is determined by the variable (column) order, which we set in $\boldsymbol{X}_k$. Thus, to minimize it, we can try to reorder the state variables, and perform an appropriate pivoting of the $\boldsymbol{J}_k$'s columns. While finding the optimal fill-reducing order is NP-complete, various heuristics – variations of the minimum degree algorithm – exist, and most prominently, `COLAMD` (Davis et al., 2004). The amount of fill-in also determines the computational cost of performing back-substitution, though this cost is normally less significant than that of the factorization itself.

QR Factorization as Variable Elimination

As we mentioned, application of the QR factorization algorithm (marked `qr`) decomposes a matrix to a product of an orthogonal matrix and an upper triangular matrix (Golub and Loan, 1996). To better understand the duality between belief and matrix factorizations, let us briefly cover a variation of this matrix factorization algorithm, which can be seen through the eyes of variable elimination.

When calculating the factorization of $\boldsymbol{J}_k$, we gradually build $\boldsymbol{R}_k$ from it, in ascending order of rows. We shall first categorize blocks of rows according to the index of their leading NZ;

that is, the column-index of the first entry on the row, in which the value is NZ. We recall that the NZ entries represent the variables involved in the respective factor; thus, we, in fact, group factors according to their first involved variable.Then, by applying an appropriate orthogonal transformation on it, each block of rows (in order) is reduced to the respective row of $\boldsymbol{R}$, representing the conditional, and possibly new rows, representing the "marginal factors"; these rows are to be processed in the next factorization steps. Each such transformation thus represents the elimination of the respective variable.

After going through all block of rows, the algorithm terminates, and returns $\boldsymbol{R}_k$. This algorithm is described in Algorithm 2, and visualized in Fig. 3.4. We use $\mathtt{I}_{\boldsymbol{J}_k}\{j_1 : j_2\}$ to mark the set of indices of rows of $\boldsymbol{J}_k$, in which the leading NZ is between the j_1-th and j_2-th columns. Note we often do not need to calculate the orthogonal matrix $\boldsymbol{Q}_k$ explicitly; instead, we can gradually apply the orthogonal transformations ($\boldsymbol{Q}_{\text{block}}$'s) directly on the RHS vector of the system, while factorizing the coefficient matrix. If still needed, we can form $\boldsymbol{Q}_k$ from its blocks through row-index tracking, or by simply calculating $\boldsymbol{Q}_k = \boldsymbol{J}_k \cdot {\boldsymbol{R}_k}^{-1}$.

Also note that to process each block of rows, we should actually perform an "internal" `qr` step, to find the appropriate orthogonal transformation. In practicality, to simply find the square root matrix, we could just apply `qr` on the entire Jacobian matrix at once (or apply $\mathtt{chol}\{\boldsymbol{\Lambda}_k\}$). However, the essence of this `qr` variation is its gradual application; this gradualness allows us to "cache" the marginal factors, and by such, return to intermediate steps in the factorization process. This is a useful property for sequential estimation problems, as we shall explain ahead.

Algorithm 2: Gradual `qr` factorization with marginal factor caching.

Inputs:
- $\boldsymbol{J} \in \mathbb{R}^{m \times n}$

Output:
- upper triangular factor $\boldsymbol{R}$
- *marginals* – a set of n matrices, each containing the marginal rows remaining after eliminating the respective column

$\boldsymbol{R} \leftarrow zeros(n, n)$; // init. matrix
marginals $\leftarrow \{\}$; // init. empty set
$Marginal \leftarrow zeros(0, n)$; // init. empty matrix
for j **to** 1 **to** n **do**
  $\{\boldsymbol{Q}_{\text{block}}, \boldsymbol{R}_{\text{block}}\} \leftarrow \mathtt{qr}\left\{\begin{bmatrix} Marginal \\ \boldsymbol{A}(\mathtt{I}_{\boldsymbol{J}}\{j\}, j : n) \end{bmatrix}\right\}$
  $\boldsymbol{R}(j, j : n) \leftarrow \boldsymbol{R}_{\text{block}}(1, 1 : end)$; // j-th row of $\boldsymbol{R}$
  $Marginal \leftarrow \boldsymbol{R}_{\text{block}}(2 : end, 2 : end)$
  $marginals\{j\} \leftarrow Marginal$

3.2.2 Incremental State Inference

According to the former explanation, incorporation of new factors can be seen as adding new linear equations to the system. Hence, to update the Jacobian matrix from time-step k, to time-step $k+1$, after performing a control u_{k+1}, and receiving an observation z_{k+1}, we need to simply add new rows to $\boldsymbol{J}_k$, such that

$$\boldsymbol{J}_{k+1} \doteq \begin{bmatrix} \boldsymbol{J}_k & \boldsymbol{0} \\ - & \boldsymbol{J}^{\delta}_{k+1} & - \end{bmatrix}. \tag{3.22}$$

Note that since controls and observations may introduce new variables to the state vector, its size at time-step k, often does not match its size at time-step $k+1$. In order for the columns

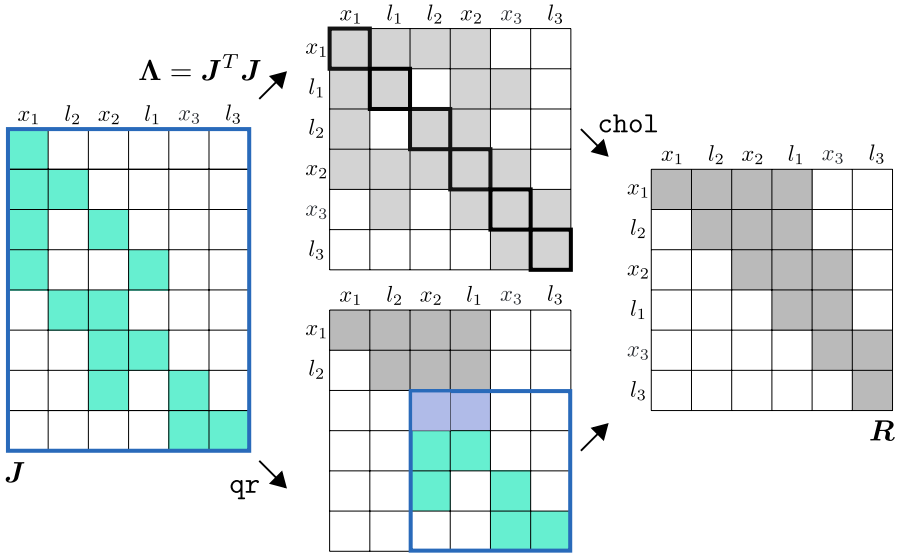


Figure 3.4: Derivation of the belief's "square root" matrix $\boldsymbol{R}$ (on the right) from its "Jacobian matrix" $\boldsymbol{J}$ (on the left). The sparsity pattern of $\boldsymbol{J}$ matches the structure of the factor graph from Fig. 3.1; the sparsity pattern of $\boldsymbol{R}$ matches the structure of the Bayes net from Fig. 3.2b. We may derive $\boldsymbol{R}$ by applying the Cholesky factorization on the information matrix $\boldsymbol{\Lambda}$ (top center). Alternatively, we may apply a "gradual" QR factorization on $\boldsymbol{J}$ directly (bottom center): the sub-matrix left for factorization is marked in blue; new "marginal" rows (in purple) may be added to the matrix in intermediate factorization steps; the sparsity pattern of the intermediate matrix matches the structure of the graph from Fig. 3.2a.

in the "old" and "new" rows to match, this addition of rows also requires proper augmentation of $\boldsymbol{J}_k$ with zero-columns, accounting for newly introduced variables $\mathcal{X}_{k+1}^{\delta}$. Adding new variables is possible at any index in the state, as long as we make sure the placement of the augmented columns is according to their index in the state vector $\boldsymbol{X}_{k+1}$; here we assumed the new variables were added to the end of the state.

Then, the belief at time-step $k+1$, which remains (approximately) Gaussian, is provided as:

$$b_{k+1}(\boldsymbol{X}_{k+1}) \approx \mathcal{N}(\boldsymbol{X}_{k+1}^{*}, \boldsymbol{\Lambda}_{k+1}^{-1}), \tag{3.23}$$

where

$$\boldsymbol{\Lambda}_{k+1} \doteq \boldsymbol{J}_{k+1}^{T}\boldsymbol{J}_{k+1} = \breve{\boldsymbol{\Lambda}}_k + \boldsymbol{J}_{k+1}^{\delta}{}^{T}\boldsymbol{J}_{k+1}^{\delta} \tag{3.24}$$

is the updated information matrix, and $\breve{\boldsymbol{\Lambda}}_k$ is the augmented prior information matrix; for example, if new variables were added to the end of the state, this matrix would be of the following form:

$$\breve{\boldsymbol{\Lambda}}_k \doteq \left(\begin{array}{c|c} \boldsymbol{\Lambda}_k & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{0} \end{array}\right). \tag{3.25}$$

To perform MAP inference, and update our estimate, we shall re-solve the updated system, by calculating the square root matrix $\boldsymbol{R}_{k+1}$ of $\boldsymbol{\Lambda}_{k+1}$. Continuing the duality between PGMs and matrices, this matrix factorization can also be computed incrementally; i.e., $\boldsymbol{R}_{k+1}$ can be efficiently calculated using $\boldsymbol{R}_k$, instead of calculating the factorization of $\boldsymbol{J}_{k+1}$ or $\boldsymbol{\Lambda}_{k+1}$ "from scratch". First, we should identify the index of the first NZ column in $\boldsymbol{J}_{k+1}^{\delta}$; we mark it as j. According to the QR algorithm, as described before, calculation of the top $j-1$ rows of $\boldsymbol{R}_k$ and $\boldsymbol{R}_{k+1}$ is equivalent, and we only need to calculate the bottom right block of $\boldsymbol{R}_{k+1}$,

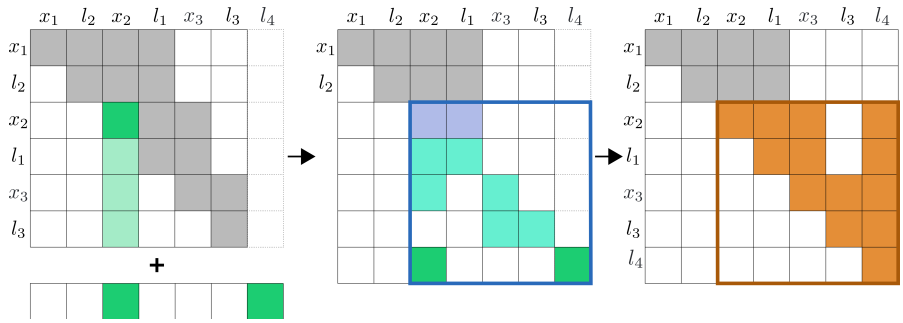


Figure 3.5: Incrementally updating the square root matrix $\boldsymbol{R}$ (top left) in the addition of new Jacobian rows (bottom left). Here, the Jacobian row has two non-zero columns (in green). We shall first recognize the first non-zero column (here, the third column). Then, we can "restart" the previous factorization process from the factorization of that column, add the new rows, and continue the process (center); the sparisity pattern of this intermediate matrix matches the structure of the graph from Fig. 3.3b. The updated square root matrix is shown on the right; the square bottom-right block, starting from the index of the first non-zero column in the update (in orange), contains the entries affected by the update. The sparsity pattern of this matrix matches the structure of the Bayes net from Fig. 3.3c.

starting from the j-th row/column (the "affected block"). Hence, given access to the marginal rows accumulated after calculating the top $j-1$ rows of $\boldsymbol{R}_k$ (i.e., elimination of the first $j-1$ variables in time-step k), we can "restart" the factorization from that intermediate stage. Then, to derive the remaining rows of $\boldsymbol{R}_{k+1}$, we can continue with factorizing these marginal rows, marked as $\boldsymbol{M}_{k,j-1}$; the remaining un-processed rows of $\boldsymbol{J}_k$ (all rows with a leading NZ $\geq j$), marked as $\boldsymbol{J}_{k,j}$; and the "new" rows $\boldsymbol{J}^{\delta}_{k+1}$. Overall, $\boldsymbol{R}_{k+1}$ is given as follows:

$$\boldsymbol{R}_{k+1} = \begin{bmatrix} & \boldsymbol{R}_k(1:j-1, 1:n_k) & \mathbf{0} \\ \mathbf{0} & - \quad \texttt{qr}(\boldsymbol{A}_{k+1}(1:end, j:end)) & - \end{bmatrix}, \text{ where } \boldsymbol{A}_{k+1} = \begin{bmatrix} \boldsymbol{M}_{k,j-1} & \mathbf{0} \\ \boldsymbol{J}_{k,j} & \mathbf{0} \\ - \quad \boldsymbol{J}^{\delta}_{k+1} & - \end{bmatrix}, \tag{3.26}$$

where `qr` marks upper triangular matrix returned by applying the QR factorization.

This idea, which is visualized in Fig. 3.5, is utilized by state-of-the-art SAM algorithm iSAM2 (Kaess et al., 2012). Again, to take advantage of such incremental updates, the order of the "unaffected" first $j-1$ variables in $\boldsymbol{X}_{k+1}$ must match the one set in $\boldsymbol{X}_k$; i.e., the order of the first $j-1$ columns of $\boldsymbol{J}_{k+1}$ must match those of $\boldsymbol{J}_k$. It is possible, and sensible, to reorder columns j to n_{k+1} of $\boldsymbol{A}_{k+1}$ before continuing the re-elimination; e.g., to apply a fill-reducing order. In this case, we also need to make sure to reorder the top $j-1$ rows of $\boldsymbol{R}_{k+1}$ accordingly. Such incremental application of fill-reducing ordering tactics is likely to be inferior, in terms of the resulting fill-in, than if we were to consider reordering of all the variables, but proves to still be effective. To clarify, it is not possible to reorder the variables in order to make the affected block (which is due to re-factorization) smaller.

Of course, this idea is only applicable if we cache the marginal rows (and the RHS vector) while calculating the prior factorization, as suggested before. An alternative method, as used in the SLAM++ algorithm (Ila et al., 2017), suggests to calculate the affected block using Cholesky (re-)factorization, and without marginal caching. It is even possible to update $\boldsymbol{R}_k$ with $\boldsymbol{J}^{\delta}_{k+1}$ directly, using "QR update", as identified in iSAM1 (Kaess et al., 2008). Though the latter two methods are arguably easier to implement, they might lead to accumulation of fill-in in $\boldsymbol{R}$, due the inability to properly utilize fill-reducing ordering during such updates. After re-factorizing the system, we shall again perform back substitution, in order to update the MAP estimate.

3.3 Planning with High-Dimensional Beliefs

Up until now, we examined a "passive" problem, in which we cared to estimate an agent's state, given known actions and observations. We now wish to actively seek the next course of action for the agent. We formulate this planning problem in compliance to the Partially Observable Markov Decision Process (POMDP) model.

Assume the agent currently has the belief b_k over its state $\mathcal{X}_k$. The goal is to select (online) the optimal policy $\pi \in \Pi$ – a mapping from beliefs to actions – which, when applied T times ("the planning horizon"), starting from b_k, would maximize the expected accumulated discounted reward ("the expected return"), as measured with the value or objective function:

$$V\left(b_k, \pi\right) \doteq \mathop{\mathbb{E}}_{\mathcal{Z}_{k+1:k+T-1}} \left[\sum_{t=1}^{T} \gamma_t \cdot \left[\rho\left(b_{k+t-1}, u_{k+t}\right) \mid \pi, \mathcal{Z}_{k+1:k+t-1} \right] \right], \tag{3.27}$$

where, for $t \geq 1$, $\mathcal{Z}_{k+1:k+t-1} \doteq \{z_{k+1}, \ldots, z_{k+t-1}\}$; $u_{k+t} = \pi(b_{k+t-1})$; and the belief b_{k+t-1} is recursively defined as in (3.11), for a given pose transition and observation models, as defined in (3.1)-(3.2). Note that for generality, we mark the discount factor $\gamma_t \in [0, 1]$ as a function of the index, to not force reward diminishment. We may also consider an arguably easier case, in which we examine predefined action sequences, instead of policies. In such case, we do not need to propagate the belief in order to determine the next action(s).

3.3.1 Reward Definition

The definition of the reward (and discount factor) must be specified by the system designer. Most often, we are interested in maximizing a state-based reward. Thus, we can define an immediate-reward function $r(\mathcal{X}_{k+t-1}, u_{t+1}, \mathcal{X}_{k+t})$ to measure the reward (scalar) of moving from state $\mathcal{X}_{k+t-1}$ to state $\mathcal{X}_{k+t}$, using u_{k+t}. Then, we can measure the expected reward of applying u_{k+t} from b_{k+t-1}:

$$\rho\left(b_{k+t-1}, u_{k+t}\right) \doteq \mathop{\mathbb{E}}_{\mathcal{X}_{k+t-1}} R(\mathcal{X}_{k+t-1}, u_{k+t}), \tag{3.28}$$

where

$$R(\mathcal{X}_{k+t-1}, u_{k+t}) \doteq \mathop{\mathbb{E}}_{\mathcal{X}_{k+t} | \mathcal{X}_{k+t-1}, u_{k+t}} r(\mathcal{X}_{k+t-1}, u_{k+t}, \mathcal{X}_{k+t}). \tag{3.29}$$

Alternatively, we may care to define a belief-based reward, and perform planning in the belief space. Then, we can define an(other) immediate-reward function r, to measure the reward of moving from belief b_{k+t-1} to b_{k+t}, using u_{k+t}, and overall measure the expected reward of applying u_{k+t} from b_{k+t-1}:

$$\begin{gathered} \rho\left(b_{k+t-1}, u_{k+t}\right) = \\ \mathop{\mathbb{E}}_{b_{k+t}} \left[r\left(b_{k+t-1}, u_{k+t}, b_{k+t}\right) \right] = \\ \mathop{\mathbb{E}}_{z_{k+t} | \mathcal{X}_{k+t}} \left[r\left(b_{k+t-1}, u_{k+t}, b_{k+t} \mid u_{k+t}, z_{k+t}\right) \right] \end{gathered} \tag{3.30}$$

Note that since for every control and/or observation, belief transition is deterministic, the expected reward is calculated by only considering the possible realizations for the observation z_{k+t}.

Specifically, in information-theoretic belief space planning (BSP), we often wish to minimize the uncertainty in the posterior belief. We can use the differential entropy as an uncertainty measure, which, for a Gaussian belief b, over a state of size n, with an information matrix

$\boldsymbol{\Lambda} = \boldsymbol{R}^T\boldsymbol{R}$, is

$$\begin{aligned}\mathrm{H}\left(b\right) &= \frac{1}{2}\cdot\ln\left[\frac{\left(2\pi e\right)^n}{\left|\boldsymbol{\Lambda}\right|}\right] \\ &= -\frac{1}{2}\cdot\left(\ln|\boldsymbol{R}|^2 - n\cdot\ln\left(2\pi e\right)\right) \\ &= -\frac{1}{2}\cdot\left(\sum_{i=1}^{n}\ln(\boldsymbol{R}_{ii})^2 - n\cdot\ln(2\pi e)\right),\end{aligned} \tag{3.31}$$

where the subscript $\square_{ij}$ marks the matrix element in the i-th row and j-th column.

We can thus define the following information-theoretic immediate-reward function:

$$r\left(b_{k+t-1}, u_{k+t}, b_{k+t}\right) \doteq \mathrm{H}(b_{k+t-1}) - \mathrm{H}(b_{k+t}), \tag{3.32}$$

which represents the information gain between time-steps. With no reward discounting, this would lead to the following objective function:

$$\boxed{\tilde{V}\left(b_k, \pi\right) \doteq \mathop{\mathbb{E}}_{\mathcal{Z}_{k+1:k+T}}\left[\mathrm{H}(b_k) - \mathrm{H}(b_{k+T} \mid \pi, \mathcal{Z}_{k+1:k+T})\right],} \tag{3.33}$$

which measures the expected information gain between the current and final beliefs.

3.3.2 Predicting Observations

Evidently, to calculate the objective function, the prior belief should be propagated (updated) according to the candidate policy. Yet, utilization of each policy may lead to different outcomes, considering the observations received during the execution. We thus seek to calculate for each candidate the expected return, considering different predicted hypotheses for the observation set. In practice, when the observation space is infinite or too large to handle, we must use an empirical estimation(s) of V for decision making, by predicting (sampling) a finite subset of hypotheses for $\mathcal{Z}_{k+1:k+T}$ (for each candidate). Surely, it is sometimes possible to approximate the return by also sampling and propagating *state* particles along each predicted trajectory; however, a "good" representation of the belief over the high-dimensional state would require an outstanding amount of such particles, making it unfeasible. We hence focus here only on parametric belief propagation, as extensively covered in the previous sections.

To appropriately predict observation realizations according to the generative model (i.e., sample them according to their likelihood), we can examine an "hypothesis tree", for every candidate policy, as seen in Fig. 3.6. When following a certain path from the root down to a leaf, the policy is incrementally executed, and control actions and observations are selected (given the latest belief) and incorporated (into that belief) in alternation. Thus, every path in this tree represents a hypothesis – an appropriate assignment to the Markov chain $H_{k+1:k+T} \doteq [u_{k+1}, z_{k+1}, \ldots, u_{k+T}, z_{k+T}]$. After "completing" each step, we can calculate its reward.

Beyond affecting the high cost of belief propagation in high dimensional states, the "curse of dimensionality" also plays an important role in cost of predicting observations. Generally, if the state is small (e.g., a single pose), or the entire state is observed each time, then we only need to contemplate about the observation value, considering the observation stochastic noise. However, when the observations may involve only a subset of the state variables, as we considered before, we actually have to ask two questions before predicting each observation: "which state variables (if any!) are observed?", and (only then) "what is the observation value?".

We thus need to "pass" two layers in the hypothesis tree, in order to sample an observation in each time-step. The branching in the first layer represents variation in the topology of the

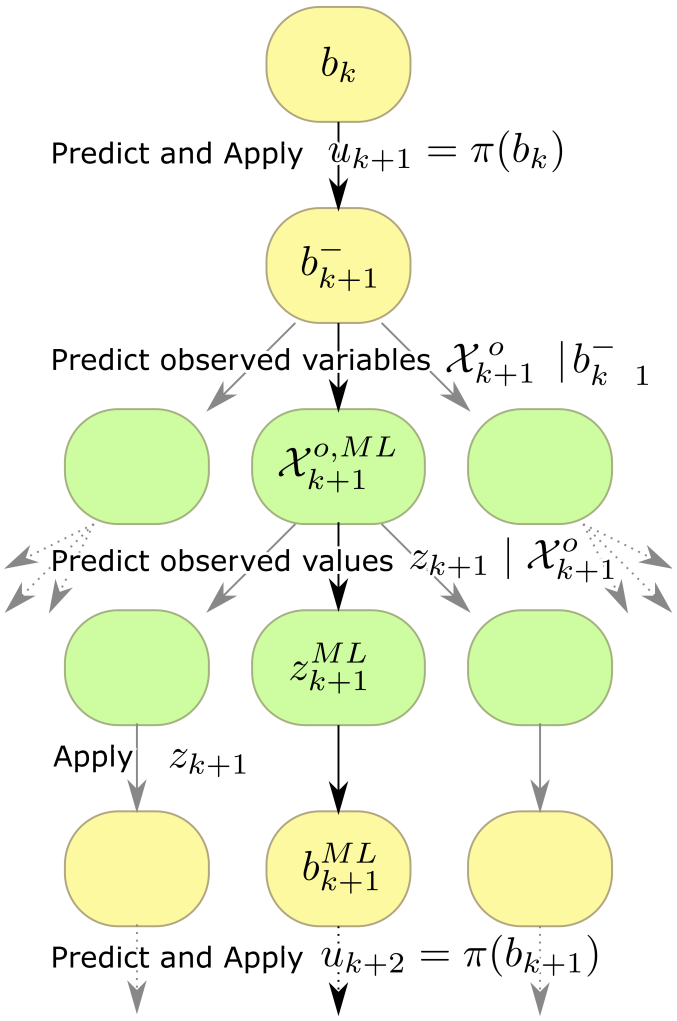


Figure 3.6: The tree of predictable hypotheses of a policy π, starting from belief b_k. Though belief transition is deterministic, the uncertainty of the future observations leads to hypothesis branching. The "maximum likelihood" hypothesis (in black) is single well-defined path in tree, which is inferable without explicit belief updates.

observation, where all observations derived from each of these branches involve the same subset of state variables; the branching in the second layer represents variation in the observed value. This double branching means that the breadth of the hypothesis tree can quickly "explode" beyond our computation ability. This is especially problematic in our case, since: (1) we want to solve the planning online; (2) for each hypothesis, we need to perform high-dimensional belief update (and not, e.g., simple particle propagation), which is computationally demanding; and (3) the process of (realistic) observation generation is typically not trivial; e.g., in vision-based navigation, we would have to calculate landmark observability from our current pose (i.e., which landmarks are in our field of view), considering multiple realizations for the state value.

Luckily, it was shown that using only a single hypothesis per candidate is often enough to achieve sufficient accuracy, by assuming "Maximum Likelihood (ML)" observations (see Platt et al. (2010)); it is hence a common assumption taken to practically solve the online planning problem. According to this assumption, we should only consider the "most likely" observation z_{k+t} in each time-step $k+t$. Such observation relies on the MAP estimate of the belief, which is, in the Gaussian case, its mean $\overline{\boldsymbol{X}}_k$. Given this realization for the state variables, we can determine which variables would be observed (involved) $\mathcal{X}^o_{k+t}$, and infer the ML observation from the observation model. When the model noise is a zero-mean Gaussian, as considered before, this observation would simply be $z^{ML}_{k+t} = h_{k+t}(\mathcal{X}^o_{k+t})$. So, the "ML observations assumption" translates every candidate policy into a specific "action and observation sequence" (i.e., a single hypothesis $H_{k+1:k+T}$ from the hypothesis tree), according to which the initial belief should be propagated.

Further, it is clear that updating the belief b^-_{k+t} (derived from the belief b_{k+t-1} after performing the action u_{k+t}, but before incorporating the observation z_{k+t}) using the ML observation z^{ML}_{k+t}, would not change its MAP estimate – since we used this exact value to generate the observation. Thus, assuming the policy function depends only on the belief's MAP estimate, the next action to be taken $u_{k+t+1} = \pi(b_{k+t})$ can be determined without calculating the updated belief b_{k+t}, but just using the MAP estimate of b^-_{k+t}. Also, updating b_{k+t}'s mean, after applying the action u_{k+t+1} (which has a zero-mean noise model), is done by simply calculating the noiseless transition function: $\overline{x_{k+t+1}} = g_k(\overline{x_{k+t}}, u_{k+t+1})$; thus, we can conveniently generate the next ML observation z^{ML}_{k+t+1}, without actually calculating the updated belief b_{k+t+1}. Overall, this means that the entire "ML hypothesis" (and the posterior MAP estimate) is known *in advance*, before performing any belief updates, and by only iteratively applying the deterministic transition and overvaluation function.

3.3.3 Problem Definition

Since solving this planning problem requires to perform belief propagation considering multiple candidates, and over long horizons, the computational cost of the solution can turn exceptionally high. The goal of this work is to help alleviate the computational cost of this problem's solution. As implied, this problem is comprised of many sub-problems, e.g., candidate action generation (or motion planning, in the case of mobile robots); proper motion and observation prediction; reward engineering; and more. Yet, in this work we focus on the specific sub-problem of *candidate comparison*, in which plainly wish to decide what is the best candidate out of a given set of candidates, and given a certain quality measure. Thus, overall, we aim to allow efficient solution to the following decision problem: given an initial belief b_k over the state $\mathcal{X}_k$, with known transition and observation models, a set Π of candidate policies, and a chosen objective function V (such as $\tilde{V}$), return the optimal candidate π^*, such that

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} V\left(b_k, \pi\right). \tag{3.34}$$

As stated before, we mark such a decision problem with the 3-tuple $\mathcal{P} \doteq (b_k, \Pi, V)$.

Chapter 4

Belief Sparsification

In Chapter 2, we examined the concept of decision problem simplification. We now wish to practically apply this idea to the decision problem $\mathcal{P} = \left(b, \Pi, \tilde{V}\right)$, which relies on the information theoretic objective (3.33), as defined at the end of Chapter 3, to allow efficient decision making in the belief space. As explained, in this domain, the "initial state" of the problem is actually a belief, and the "candidate actions" are actually control policies for the agent. Here, we choose keep the same objective function and set of candidates, and focus on simplifying the initial belief.

To evaluate the candidates and solve this problem, we are inherently required to perform *multiple updates* on the (same) initial belief "in parallel", propagating it according to various predicted hypotheses. Hence, let $b(\boldsymbol{X})$ mark the initial ("prior") belief, and $FG \doteq (\mathcal{X} \cup \mathcal{F}, \mathcal{E})$ mark its corresponding factor graph; in the Gaussian case, this belief is represented with the mean vector $\boldsymbol{X}^*$, and information matrix $\boldsymbol{\Lambda} \doteq \boldsymbol{J}^T\boldsymbol{J}$, such that $b \approx \mathcal{N}(\boldsymbol{X}^*, \boldsymbol{\Lambda})$. Also, let $\mathcal{H} \doteq \{H_1, \ldots, H_m\}$ represent the set of all predicted hypotheses (of all candidates); in the BSP case, as we defined before, the hypotheses are of the form $H \doteq [a_1, z_1, \ldots, a_t, z_t]$. For the coherency of the discussion, in this chapter, one can assume that each candidate policy corresponds to *one* of the predefined hypotheses in $\mathcal{H}$, e.g., by utilizing the "ML observations" assumption; yet, this assumption is not essential to our suggested approach(es). This planning scenario is later visualized in Fig. 4.5. Each hypothesis H can be matched with its own factor graph $FG^\delta_H \doteq (\mathcal{X} \cup \mathcal{X}^\delta_H \cup \mathcal{F}^\delta_H, \mathcal{E}^\delta_H)$ ("update"), containing the variables and constraints we wish to integrate into the initial belief, and with which we wish to extend the initial factor graph, such that $FG_H \doteq FG \cup FG^\delta_H$; in the Gaussian case, each update H is matched with a matrix $\boldsymbol{J}^\delta_H$ containing the new rows with which we want to extend the matrix $\boldsymbol{J}$; we refer to this matrix as the "collective Jacobian" of the hypothesis. The posterior belief, after performing the respective update, is marked b_H.

We should note that in many problems, especially in navigation problems, the collective Jacobians are inherently sparse, and as the state grows, involve less variables in relation to its size. Hence, we may conclude that the cost of belief update depends mainly on the density (or sparsity) of the initial belief, and shall be significantly lower for sparse ones. Thus, we suggest to simplify the decision problem by considering a sparse approximation b_s of the initial belief b. To clarify, when referring to a "sparse" belief, we mean that it is represented with a sparser factor graph, i.e., which contains less edges (and factors), in comparison to the original one; or equivalently, in the Gaussian case, a sparser information matrix (and square root information matrix), i.e., which contains less NZ entries.

Fig. 4.1 summarizes the paradigm of belief sparsification for efficient decision making in the belief space; clarification regarding its steps is to follow.

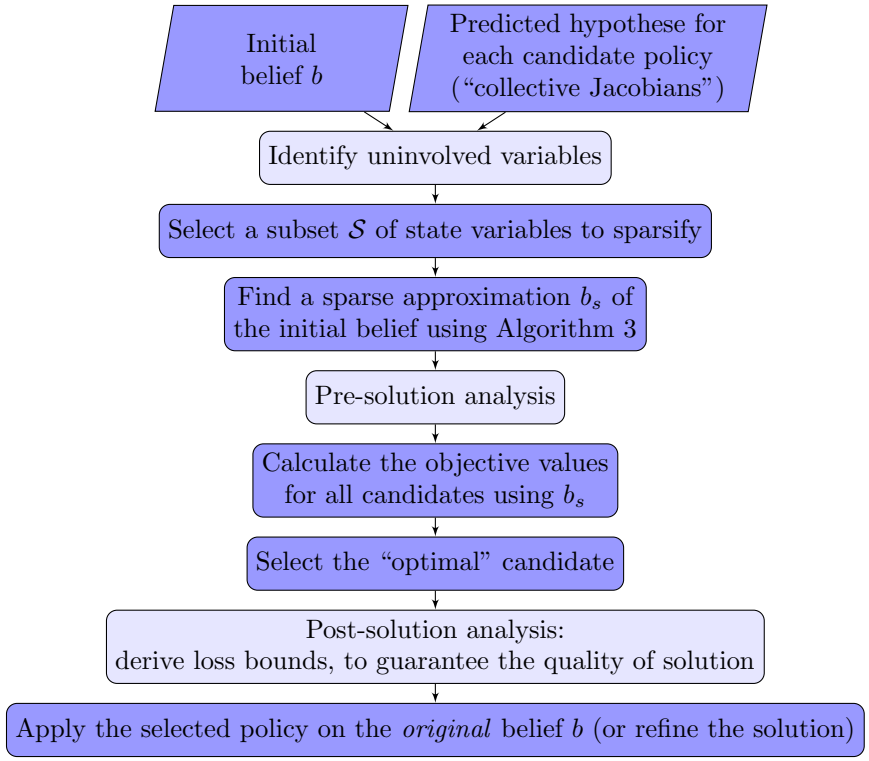


Figure 4.1: Belief sparsification for efficient decision making in the belief space. Essential steps are in dark blue; optional steps, in order to provide guarantees, are in light blue. Here, candidate actions represent control policies for the agent.

4.1 The Sparsification Algorithm: Gaussian Beliefs

For practical reasons, we start by providing an algorithm for sparsification of a *Gaussian* Belief (Algorithm 3). The algorithm may receive as input, and return as output, a belief represented using either the information matrix, or its square root. This scalable algorithm depends on a pre-selected subset $\mathcal{S}$ of state variables, and wisely removes elements which correspond to these variables from the matrix. Approximations of different degrees can be generated using different variable selections $\mathcal{S}$, as to be explained in Section 4.3.1. For a clear discussion, when $\mathcal{S}$ contains all the variables, we say this is a *full sparsification*; using any other partial selection of variables is a *partial sparsification*. Fig. 4.2 contains a visual demonstration of the algorithm steps. In the following section (Sec. 4.2), we provide an extended probabilistic analysis of the algorithm, using PGMs, and explain how it can also be applied to general (non-Gaussian) beliefs.

Let us break down the algorithm steps:

First, we should check if the variables are ordered properly, i.e., such that the variables we wish to sparsify (variables in $\mathcal{S}$) appear first in the state. If not, we should reorder the variables accordingly. This requires appropriate modification of the input matrix. If the algorithm input is the symmetric matrix $\mathbf{\Lambda}$ (line 2), we shall simply permute its rows and columns by calculating the product $\boldsymbol{P}^T\mathbf{\Lambda}\boldsymbol{P}$ of the information matrix with an appropriate (column) permutation matrix $\boldsymbol{P}$[1] (line 3). After this permutation, we can derive $\boldsymbol{R}^p$, the square root matrix of the permuted

[1] Formal definition of the permutation matrix appears in Section 6.1.

Algorithm 3: Scalable belief sparsification.

Inputs:
- A belief $b = \mathcal{N}(\boldsymbol{X}^*, \boldsymbol{\Lambda}^{-1})$, such that $\boldsymbol{\Lambda} = \boldsymbol{R}^T\boldsymbol{R}$
- A subset $\mathcal{S}$ of state variables to sparsify

Output:
- A sparsified belief $b_s \doteq \mathcal{N}(\boldsymbol{X}^*, \boldsymbol{\Lambda}_s^{-1})$, such that $\boldsymbol{\Lambda}_s \doteq \boldsymbol{R}_s^T\boldsymbol{R}_s$

`// reorder the state variables such that the variables in` $\mathcal{S}$ `are first in the state vector`

1 $\boldsymbol{P} \leftarrow$ an appropriate (column) permutation matrix
2 **if** *the algorithm input is* $\boldsymbol{\Lambda}$ **then**
3 $\quad\boldsymbol{\Lambda}^p \leftarrow \boldsymbol{P}^T\boldsymbol{\Lambda}\boldsymbol{P}$
4 $\quad\boldsymbol{R}^p \leftarrow \texttt{chol}(\boldsymbol{\Lambda}^p)$
5 **else if** *the algorithm input is* $\boldsymbol{R}$ **then**
6 $\quad\boldsymbol{R}^p \leftarrow$ modify $\boldsymbol{R}$ to convey appropriate variable reordering, e.g., by applying Algorithm 6
7 $\boldsymbol{R}_s^p \leftarrow$ zero off-diagonal elements from $\boldsymbol{R}^p$ in rows matching variables in $\mathcal{S}$ `// sparsify` $\boldsymbol{R}^p$
8 $\boldsymbol{R}_s \leftarrow \boldsymbol{P}\boldsymbol{R}_s^p\boldsymbol{P}^T$ `// return to the original variable order`
9 **if** *the algorithm output is* $\boldsymbol{\Lambda}$ **then**
10 $\quad\boldsymbol{\Lambda}_s \leftarrow \boldsymbol{R}_s^T\boldsymbol{R}_s$ `// reform the information matrix`

information matrix, using the Cholesky decomposition (line 4). If the algorithm input is the matrix $\boldsymbol{R}$ (line 5), the task of variable reordering is not trivial, as trying to modify $\boldsymbol{R}$ by permuting its rows and columns would break its triangular shape. Instead, this task (typically) requires re-factorization of $\boldsymbol{\Lambda}$ (or $\boldsymbol{J}$) under the new variable order. Fortunately, in Chapter 6 we provide an efficient modification algorithm for $\boldsymbol{R}$ (Algorithm 6), which is intended for this exact task, and can spare the matrix re-factorization; we can use it to derive $\boldsymbol{R}^p$ (line 6). Note that if needed, the "re-factorization" variant of that algorithm also allow us to properly modify the marginal factor cache, which can be used when updating the modified matrix, as explained in Section 3.2.2.

If no reordering is required, and the algorithm input is $\boldsymbol{\Lambda}$, we may directly calculate the Cholesky decomposition (line 4); if no reordering is required, and the input is $\boldsymbol{R}$, we may skip directly to line 7. Specifically, when all of $\mathcal{S}$ is already at the beginning of the state, no reordering is needed. This situation particularly occurs when sparsifying *all* the variables (i.e., full sparsification). Next, in line 7, we zero off-diagonal elements in the permuted square root matrix $\boldsymbol{R}^p$, in rows corresponding to variables in $\mathcal{S}$, to yield the sparsified square root matrix $\boldsymbol{R}_s^p$.

Since the prior belief should be updated according to the predicted hypotheses, the variable order in the sparsified information matrix (or its square root) must match the variable order in the collective Jacobians. Thus, we should reorder the variables back to their original order (line 8). Though, we notice that after the sparsification this permutation can be performed on the square root matrix *directly*, without resorting to the information matrix, and without breaking its triangular shape, by calculating $\boldsymbol{P}\boldsymbol{R}_s\boldsymbol{P}^T$ (note the reverse multiplication order). This claim is formalized in Corollary 1 (and proved in Appendix B).

Corollary 1. *After sparsification of the square root matrix (line 7 of Algorithm 3), permutation of the variables back to their original order can be performed on the square root matrix directly, without breaking its triangular shape.*

Finally, we may return the sparsified belief, represented either with $\boldsymbol{R}_s$ or $\boldsymbol{\Lambda}_s$. In the latter case, this requires to (easily) reconstruct the sparsified information matrix from its sparsified root (line 10). After the sparsification, the value of the NZ entries in the sparsified information matrix may be different than the corresponding entries in the original matrix (including the

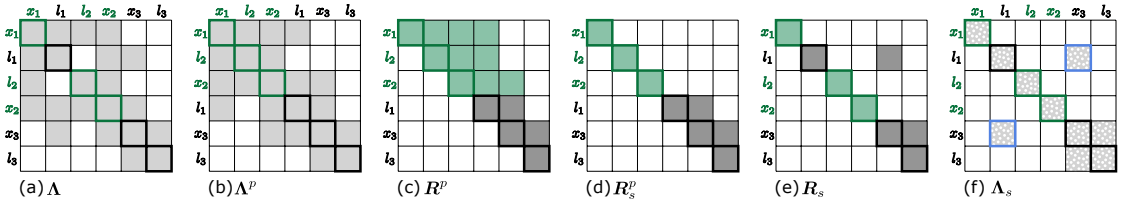


Figure 4.2: The steps of Algorithm 3 (from left-to-right), for sparsification of a Gaussian belief (shown in Fig. 4.3a); the state variables are $\boldsymbol{X} \doteq [x_1, l_1, l_2, x_2, x_3, l_3]^T$ (in that order), and the subset of variables selected for sparsification is $\mathcal{S} = \{x_1, l_2, x_2\}$ (in green). (a) The sparsity pattern of the symmetric information matrix of belief. (b) Reordering the variables, such that all the variables in $\mathcal{S}$ appear first; this is done by simply permuting the rows and columns of the matrix. (c) Calculating the upper triangular square root matrix $\texttt{chol}(\boldsymbol{\Lambda}^p)$ of the permuted information matrix; each row corresponds to a state variable. (d) Removing off-diagonal elements from rows corresponding to variables in $\mathcal{S}$. (e) After the sparsification, we may permute the variables back to their original order directly in the square root matrix, without breaking its upper triangular shape. (f) Reforming the sparsified information matrix $\boldsymbol{\Lambda}_s \doteq \boldsymbol{R}_s^T \boldsymbol{R}_s$; note that the process affects the values in the matrix, and may also introduce new non-zeros (marked in purple).

diagonal), and new NZs may be added in compensation for the removed entries (factors). Also, note that the permutation of variables back to their original order can potentially be skipped, by equivalently permuting the columns of all the candidate collective Jacobians, to match the altered order.

The derivation of $\boldsymbol{R}_p$ (in line 4 or line 6), when conducted, is the costliest step of the algorithm, which defines its maximal computational complexity; we may recall that the complexity of the Cholesky decomposition is $O(n^3)$, at worst, where n is the state size (Hämmerlin and Hoffmann, 2012). In comparison, the computational cost of the remaining steps, i.e., matrix permutation (lines 3 and 8), removal of matrix elements (line 7), and reconstruction of the information matrix (line 10), is usually minor. Still, it should be noted that depending on the configuration, many of the steps are often not necessary. For example, as mentioned, when the input matrix is already in the desired order, the permutations can be skipped; this is specifically correct in full sparsification. In that case, if given the square root matrix as input, the algorithm holds an almost negligible complexity – we only need to extract the matrix' diagonal. Also, in full sparsification, the sparsified information matrix, if required, can be reconstructed from its root in linear complexity, as both $\boldsymbol{R}_s$ and $\boldsymbol{\Lambda}_s$ are diagonal.

Nonetheless, we remind that the approach is meant to overall reduce the decision making time, as the time spent on performing the sparsification (performed once) is lower than the time saved in performing (the multiple) belief updates. For example, since full sparsification leads to a diagonal approximation (information or its root), considering the collective Jacobians are sparse, belief updates can be performed with an almost linear complexity. Also, since the cost of sparsification does not depend on the number of candidates or hypotheses, as this number grows, the relative "investment" in calculating the sparsification becomes less significant.

Fig. 4.4 shows a comparison of an information root matrix (taken from our experimental evaluation, as to be seen) and its sparse approximations generated with Algorithm 3, for different variable selections $\mathcal{S}$.

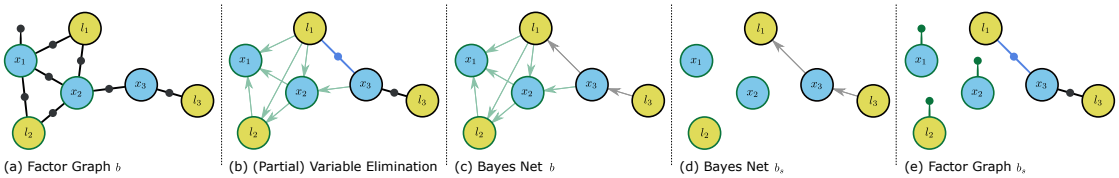


Figure 4.3: Visualizing the steps of Algorithm 3 (from left-to-right), for sparsification of a general belief with probabilistic graphical models. (a) The factor graph of the prior belief b (matching Fig. 4.2a); the state variables are $\boldsymbol{X} \doteq [x_1, l_1, l_2, x_2, x_3, l_3]^T$, and the subset of variables selected for sparsification is $\mathcal{S} = \{x_1, l_2, x_2\}$ (circled in green). (b) Eliminating the variables in the factor graph in order to derive the corresponding Bayes net; the figure describes an intermediate step of the elimination process, after eliminating the variables in $\mathcal{S}$: x_1, l_2, x_2 (in this order); note the added marginal factor (in purple). (c) The final Bayes net of b, after eliminating all the variables. (d) Removing all edges which lead to variables in $\mathcal{S}$ (green arrows); this is the Bayes net describing the sparsified belief b_s. (e) Reforming the factor graph of the sparsified belief b_s; variables in $\mathcal{S}$ are now independent, and each one is connected to a modified prior factor (in green); the remaining variables are inter-connected with the same factors which connected them originally (in black), alongside the marginal factors, which were added after elimination of $\mathcal{S}$ (in purple).

4.2 The Sparsification Algorithm: Probabilistic Analysis

Let us analyze the suggested sparsification approach from a wider perspective, using probabilistic graphical models, and explain its implications on general, and not necessarily Gaussian, beliefs.

As explained, a belief is constructed as a product of factors – probabilistic constraints between variables, and can be graphically represented with a factor graph. In Fig. 4.3a, we can see an exemplary factor graph (the same one brought in Fig. 3.1), which represents a belief b with six variables and eight factors:

$$b(\boldsymbol{X}) \propto f_{x_1} \cdot f_{x_1 l_1} \cdot f_{x_1 l_2} \cdot f_{x_1 x_2} \cdot f_{x_2 l_1} \cdot f_{x_2 l_2} \cdot f_{x_2 x_3} \cdot f_{x_3 l_3}, \tag{4.1}$$

where the state $\boldsymbol{X} \doteq [x_1, l_1, l_2, x_2, x_3, l_3]^T$ contains three poses and three landmarks, and f_{ij} is a factor between i and j. In our example, we chose the variables for sparsification to be $\mathcal{S} = \{x_1, l_2, x_2\}$. In the linear(ized) Gaussian system, the belief b can be described with an information matrix $\boldsymbol{\Lambda}$, as shown in Fig. 4.2a. Off-diagonal non-zero entries in the information matrix indicate the existence of factors between the corresponding variables. As further explained, the belief b can be factorized to a product of conditional probability distributions, through "variable elimination" (see (3.10)):

$$b(\boldsymbol{X}) \propto \prod_{i=1}^{5} \mathbb{P}(\boldsymbol{X}(i) \mid d(\boldsymbol{X}(i))) \cdot \mathbb{P}(\boldsymbol{X}(6)), \tag{4.2}$$

where $d(x)$ denotes the set of variables x depends on. Graphically, this elimination transforms the factor graph into a directed Bayes net, as shown in Figs. 4.3b-4.3c.

We want to start the sparsification process with elimination of the variables in $\mathcal{S}$. By doing so, we would force conditional separation of the variables for sparsification and the remaining variables $\neg\mathcal{S}$, i.e.,

$$b(\boldsymbol{X}) \propto \mathbb{P}(\mathcal{S} \mid \neg\mathcal{S}) \cdot \mathbb{P}(\neg\mathcal{S}). \tag{4.3}$$

This means that no variable in $\neg\mathcal{S}$ would be conditionally dependent on a variable in $\mathcal{S}$. Let us look again at our exemplary belief (4.1), and practically perform this elimination. First, x_1:

$$b \propto \mathbb{P}(x_1 \mid x_2, l_1, l_2) \cdot f'_{x_2 l_1 l_2} \cdot f_{x_2 l_1} \cdot f_{x_2 l_2} \cdot f_{x_2 x_3} \cdot f_{x_3 l_3}. \tag{4.4}$$

Then, l_2:

$$b \propto \mathbb{P}(x_1 \mid x_2, l_1, l_2) \cdot \mathbb{P}(l_2 \mid x_2, l_1) \cdot f'_{x_2,l_1} \cdot f_{x_2 l_1} \cdot f_{x_2 x_3} \cdot f_{x_3 l_3}. \tag{4.5}$$

Finally, x_2:

$$b \propto \mathbb{P}(x_1 \mid x_2, l_1, l_2) \cdot \mathbb{P}(l_2 \mid x_2, l_1) \cdot \mathbb{P}(x_2 \mid l_1, x_3) \cdot f'_{x_3 l_1} \cdot f_{x_3 l_3}. \tag{4.6}$$

This partial elimination is visualized in Fig. 4.3b (and continued Fig. 4.3c). As we can see, after elimination of variables, new "marginal" factors ($f'_{x_2 l_1 l_2}, f'_{x_2,l_1}, f'_{x_3 l_1}$) may be introduced to the belief, representing new links among the non-eliminated variables; in our case, after eliminating all the sparsified variables, one marginal factor still remains: $f'_{x_3 l_1}$.

Practically, in the linear(ized) Gaussian system, variable elimination is equivalent to factorization of $\boldsymbol{\Lambda}$ to its square root, and the elimination order is conveyed by the order of variables in the state vector $\boldsymbol{X}$. Hence why, according to Algorithm 3, we start the sparsification process by reordering the state variables, such that all variables in $\mathcal{S}$ appear first; this step requires us to permute the information matrix accordingly (as shown in Fig. 4.2b). After this permutation, we can perform a factorization of the (permuted) information matrix $\boldsymbol{\Lambda}^p$ to its square root $\boldsymbol{R}^p$ (Fig. 4.2c), whose sparsity pattern matches the structure of the eliminated belief's Bayes net.

According to the next step in the algorithm, we shall now zero off-diagonal entries in $\boldsymbol{R}^p$, in the rows which correspond to variables in $\mathcal{S}$ (Fig. 4.2d). We recall that the i-th row of this square root matrix corresponds to the i-th conditional factor in the factorization of b (the conditional probability distribution of the i-th variable): if the off diagonal entry $\boldsymbol{R}^p_{ij}$ is non-zero, then $\boldsymbol{X}(j) \in d(\boldsymbol{X}(i))$, and $\boldsymbol{X}(j)$ is a parent of $\boldsymbol{X}(i)$ in the Bayes net. Thus, this algorithm step can more generally be seen as removing edges from the Bayes net (Fig. 4.3d).

By removing all the off-diagonal entries from the i-th row, we replace the conditional probability distribution with an independent probability distribution over $\boldsymbol{X}(i)$. In the Gaussian case, this means replacing the factor

$$\mathbb{P}(\boldsymbol{X}(i)|d(\boldsymbol{X}(i))) = \mathcal{N}\left(\mu(d(\boldsymbol{X}(i))), ({\boldsymbol{R}^p_{ii}}^T \boldsymbol{R}^p_{ii})^{-1}\right) \tag{4.7}$$

with

$$\mathbb{P}_s(\boldsymbol{X}(i)) \doteq \mathcal{N}\left(\mu_i, ({\boldsymbol{R}^p_{ii}}^T \boldsymbol{R}^p_{ii})^{-1}\right). \tag{4.8}$$

Essentially, we fix the MAP estimate (i.e., mean) of $\boldsymbol{X}(i)$ to a constant value, which is no longer dependent on other variables. We, of course, would like to preserve the estimate of the overall belief, and therefore shall select $\mu_i = \boldsymbol{X}^*_i$. It should be mentioned that this independent probability distribution is *not* the marginal distribution over $\boldsymbol{X}(i)$, which is given as $\mathcal{N}(\boldsymbol{X}^*(i), \Sigma_{ii})$.

The sparisified belief is thus given as the product

$$b_s(\boldsymbol{X}) \propto \prod_{x \in \mathcal{S}} \mathbb{P}_s(x) \cdot \mathbb{P}(\neg \mathcal{S}). \tag{4.9}$$

The chosen elimination order makes sure that the inner dependencies among the non-sparsified variables remain exact. Notably, the suggested sparsification is performed by manipulating the the Bayes net, in contrast to traditional belief sparsification methods (as we reviewed), which perform sparsification directly on the factor graph.

Still, to understand how the factor graph of our sparsified belief looks like, let us consider again our exemplary belief after performing partial elimination (4.6). According to the previous analysis, in the sparsification, each of the conditional distributions on the sparsified variables is replaced with an independent distribution. These are, in fact, unitary factors over the variables; here, we mark those as f''_{x_1}, f''_{l_1}, f''_{x_2}. The sparsified belief can thus be given as a product of these unitary factors on the sparsified variables, the marginal factors introduced after eliminating these

variables, and the remaining non-eliminated factors (here, $f_{x_3 l_3}$). Overall, in our example, this product is:

$$b_s \propto f''_{x_1} \cdot f''_{l_1} \cdot f''_{x_2} \cdot f'_{x_3 l_1} \cdot f_{x_3 l_3} \tag{4.10}$$

The factor graph matching this belief is shown in Fig. 4.3e; the matching sparsified information matrix, which can be reconstructed from its sparsified root, is shown in Fig. 4.2f. It is hence clear that the sparsification does not affect the elimination of the remaining variables (variables in $\neg\mathcal{S}$). Continuing the elimination process from either b (4.6) or b_s (4.10) would result in the same distribution $\mathbb{P}(\neg\mathcal{S})$.

To complete the analysis, we shall note that this sparsification method does not change the diagonal entries in the square root matrix, and, thus, the determinants of $\mathbf{\Lambda}$ and $\mathbf{\Lambda}_s$ remain the same:

$$|\mathbf{\Lambda}| = |\mathbf{\Lambda}^p| = \left|\boldsymbol{R}^{p\,T}\boldsymbol{R}^p\right| = |\boldsymbol{R}^p|^2 = \prod_i^n (\boldsymbol{R}^p_{ii})^2 = |\boldsymbol{R}^p_s|^2 = \left|\boldsymbol{R}^{p\,T}_s \boldsymbol{R}^p_s\right| = |\mathbf{\Lambda}^p_s| = |\mathbf{\Lambda}_s|. \tag{4.11}$$

Hence, the sparsification method preserves the overall entropy of the belief (3.31), no matter which variables are sparsified. This is usually not guaranteed in the aforementioned traditional sparsification methods. Still, when incorporating new factors in the future, divergence in entropy between the original and sparsified beliefs (i.e., offset) might indeed happen. This offset depends on the variables selected for sparsification, and can even be zero, as we shall discuss next. Since the sparsified variables become independent, if we wish to update our estimation after applying new actions, or after acquiring a new observation of an existing variable (i.e., loop closure), information will no longer propagate from a sparsified variable to another variable, or vice-versa, unless they are observed together. Though, notably, unlike simply marginalizing the sparsified variables out of state, as done in filtering, they can still be updated in the future.

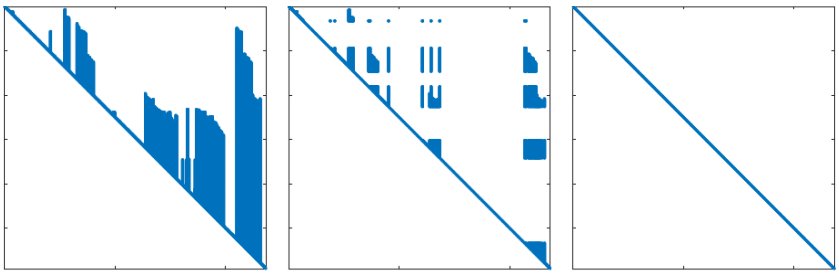


Figure 4.4: A square root matrix (taken from our experimental evaluation) and its sparse approximations generated with Algorithm 3, for different variable selections $\mathcal{S}$. On the left – the original matrix; in the center – the matrix after partial sparsification, of only the uninvolved variables (here, about half of the variables); on the right – the matrix after full sparsification, which results in a convenient diagonal approximation. The beliefs represented by the matrices on the left and in the center are guaranteed to be action consistent.

4.3 Optimality Guarantees

4.3.1 Variable Selection and Pre-Solution Guarantees

Next, we wish to present conclusions derived from symbolic evaluation of this simplification method, as we previously suggested in our theoretical framework for simplified decision making, in Chapter 2. In this evaluation, we utilized our knowledge on the decision problem formulation, and on Algorithm 3, in order to derive general guarantees for the simplification loss.

Variable Selection

We remind again that according to Lemmas 1 and 2 (in Section 2.2), we can use (a bound of) the offset between the problem and its simplification, to bound the induced loss. Thus, we shall begin by explaining which variables should be sparsified, such that the effect on the objective value for each candidate (i.e., the simplification offset) is minimal.

As previously defined, the involved variables in each hypothesis (update) H are defined as the variables which are connected to at least one of the factors introduced by this update $\mathcal{F}_H^\delta$. In the Gaussian case, these are identified by NZ columns in the collective Jacobian matrix J_H^δ, where each of the columns corresponds to a state variable, and every row represents a constraint; a variable is involved if at least one of the entries in its matching column is NZ. For example, in a navigation scenario, the landmarks we predict to observe by taking the action (along with the current pose) are involved; variables referring to landmarks from the past, which we do not predict to observe, are uninvolved. An illustration of this example is given in Fig. 4.5.

By definition, all the newly added variables in the update are necessarily involved; yet, in our analysis, we only care for variables which are a part of the prior state, i.e. $\subseteq \mathcal{X}$. For each hypothesis H, we mark the subset of involved prior-state variables as $\mathcal{I}nv_{\mathcal{X}}(H)$. The remaining variables are uninvolved in this update, and marked $\neg\mathcal{I}nv_{\mathcal{X}}(H)$. Accordingly, the variables which are involved in at least one update $H \in \mathcal{H}$ are marked $\mathcal{I}nv_{\mathcal{X}}(\mathcal{H})$, and the variables which are uninvolved in any of the updates are marked $\neg\mathcal{I}nv_{\mathcal{X}}(\mathcal{H})$. We use the subscript $\boldsymbol{X}$, instead of $\mathcal{X}$, to refer to the respective subset of variables, *ordered* according to the variables' index in $\boldsymbol{X}$.

We claim that for any predicted hypothesis, sparsifying its uninvolved variables from the prior belief b, before updating it, does not affect the posterior entropy (which defines our objective function $\tilde{V}$). Hence, for a set of candidate policies Π, we can sparsify from the prior belief all variables which are uninvolved in any of the hypotheses predicted (for the candidates), and use this sparsified belief b_s to compute the objective function, without affecting its values. Specifically, this means that the simplification offset is zero, and that this sparsified belief is action consistent with the original one: $b \simeq b_s$. This claim is formally expressed in Theorem 1. A proof for this claim is given in Appendix B.

Theorem 1. *Consider a decision problem* $\mathcal{P} \doteq \left(b, \Pi, \tilde{V}\right)$*, where* b *is a (Gaussian) initial belief, and* $\tilde{V}$ *is the information-theoretic objective function from (3.33). Considering a set* $\mathcal{S}$ *of state variables, which are uninvolved in any of the (hypotheses predicted for the) candidates in* Π*, Algorithm 3 returns a belief* b_s*, such that* $\Delta(\mathcal{P}, \mathcal{P}_s) = 0$*, where* $\mathcal{P}_s \doteq \left(b_s, \Pi, \tilde{V}\right)$*.*

Since sparsifying uninvolved variables does not affect the objective values, when they can be identified, they should always be included in the set $\mathcal{S}$ of variables for sparsification. As explained in Section 3.3.2, when assuming ML observations, the entire predicted hypotheses, and, by such, the variable classification (un/involved) can be determined in advance. We emphasize that since this is a planning problem, the variable classification is *determined* based on our prediction, which can only be based on our current belief, and not the ground truth (as it is unknown).

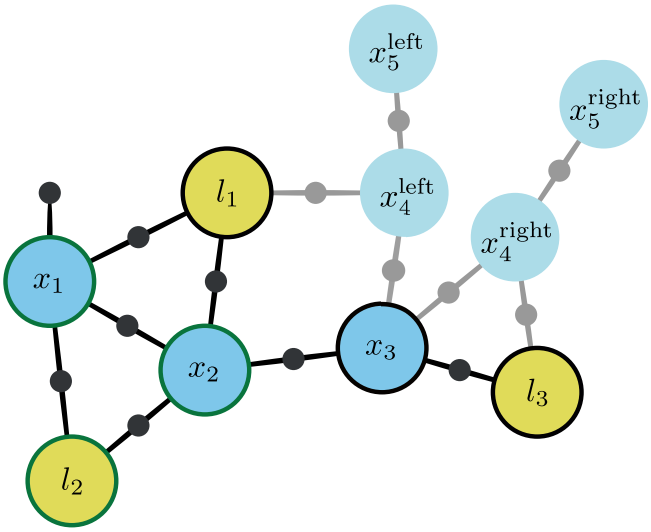


Figure 4.5: An exemplary factor graph representing an "active-SLAM" planning scenario. The factor graph from Fig. 3.1 represents the initial belief. At time of planning, the agent is at pose x_3, and wishes to infer which of the candidate paths $\Pi = \{\text{left, right}\}$ is the optimal one. For each candidate we predict a certain hypothesis for the future. If taking the right path, the agent predicts augmenting its state with two new poses $x_4^{\text{right}}, x_5^{\text{right}}$, with motion factors connecting them to the current pose; based on its current state estimation, it also predicts observing landmark l_3 from x_4^{right} (i.e., adding an observation factor between l_3 and the new pose). The variables (from the prior state) involved with this action are those directly connected to any of the predicted new factors – x_3, l_3. If taking the left path, the agent predicts augmenting its state with two new poses $x_4^{\text{left}}, x_5^{\text{left}}$, and observing landmark l_1 from x_4^{left}. The variables involved with this action are x_3, l_1. The involved variables (in any of the actions) are marked with black circles. Note that x_1, x_2, l_2 are never involved; these are marked with dark green circles. To solve the problem, we shall update the initial belief, according to each of these hypotheses.

In general, only a single sparsification process should be conducted for each decision problem (i.e., planning session), regardless of the number of candidate actions. Selecting variables which are uninvolved in any of the hypotheses allows to keep action consistency considering the entire set of candidates. Still, it is possible to break the set of hypotheses to several subsets of similar ones, and consider the uninvolved variables in each subset. For each subset we would create a custom prior approximation, and use it to when updating the belief according to that hypothesis. This can result in a more adapted sparsification for each subset. Yet, calculation of the sparsification itself has a cost, which needs to be considered when trying to achieve the best performance.

Surely, we may consider multiple predicted hypotheses for each candidate, each determining its own set of involved variables. However, in that case, as we recall, the prediction and variable classification might not be fully accessible in advance. Then, if we wish to still rely on Theorem 1, and guarantee action consistency, it is possible to consider a "gradual" simplification, as the candidate policies are incrementally evaluated. This approach is explained in depth later in this thesis, in the context of another suggested simplification method – `PIVOT` (in Section 5.2). Nonetheless, it is surely possible to forfeit this "zero offset" guarantee, and sparsify involved variables, or variables with an undetermined classification. Intuitively, selecting more involved variables to $\mathcal{S}$ results in a sparser approximation, and improved performance, but potentially a larger divergence (offset) from the original objective values.

Pre-Solution Guarantees for Rank-1 Updates

We proved that sparsification of the uninvolved variables always results in zero offset, and hence zero loss. Now, we show that under additional restrictions, we can symbolically derive offset (and loss) bounds also when sparsifying involved variables; these bounds are only applicable for "rank-1" updates, i.e., when the updates' collective Jacobians are limited to a single row.

Assume that for a candidate $\pi \in \Pi$ we derived a (single) hypothesis, for which the corresponding collective Jacobian is the row vector $\boldsymbol{U} \in \mathbb{R}^{1\times N}$. This can be the case, for example, in sensor placement problems with scalar measurements (like temperature). Now, let us analyze the simplification offset[2]:

$$\delta(\mathcal{P}, \mathcal{P}_s, \pi) = \tag{4.12}$$

$$\left|\tilde{V}(b,\pi) - \tilde{V}(b_s,\pi)\right| = \tag{4.13}$$

$$\left|\ln\left|\boldsymbol{\Lambda} + \boldsymbol{U}^T\boldsymbol{U}\right| - \ln\left|\boldsymbol{\Lambda}_s + \boldsymbol{U}^T\boldsymbol{U}\right|\right| = \tag{4.14}$$

(Matrix determinant lemma (see Harville, 1998))

$$\left|\ln\left(|\boldsymbol{\Lambda}|\cdot\left(1+\boldsymbol{U}\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^T\right)\right) - \ln\left(|\boldsymbol{\Lambda}_s|\cdot\left(1+\boldsymbol{U}\boldsymbol{\Lambda}_s^{-1}\boldsymbol{U}^T\right)\right)\right| = \tag{4.15}$$

(from (4.11))

$$\left|\ln\left(1+\boldsymbol{U}\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^T\right) - \ln\left(1+\boldsymbol{U}\boldsymbol{\Lambda}_s^{-1}\boldsymbol{U}^T\right)\right| = \tag{4.16}$$

$$\left|\ln\left(1+\boldsymbol{U}\boldsymbol{\Lambda}_s^{-1}\boldsymbol{U}^T + \boldsymbol{U}(\boldsymbol{\Lambda}^{-1}-\boldsymbol{\Lambda}_s^{-1})\boldsymbol{U}^T\right) - \ln\left(1+\boldsymbol{U}\boldsymbol{\Lambda}_s^{-1}\boldsymbol{U}^T\right)\right| = (\star) \tag{4.17}$$

The logarithm is a monotonously increasing concave function; thus, every $a, b \in \mathbb{R}$ and $c \geq 0$ satisfy

$$|\ln(a) - \ln(b)| \geq |\ln(a+c) - \ln(b+c)|. \tag{4.18}$$

In other words, the difference in the function value between a pair of inputs decreases, when the inputs equally grow. Surely, $0 \leq \boldsymbol{U}\boldsymbol{\Lambda}_s^{-1}\boldsymbol{U}^T$, since $\boldsymbol{\Lambda}_s^{-1}$ is positive semi-definite. Thus, we may choose $a = 1 + \boldsymbol{U}(\boldsymbol{\Lambda}^{-1} - \boldsymbol{\Lambda}_s^{-1})\boldsymbol{U}^T$, $b = 1$, and $c = \boldsymbol{U}\boldsymbol{\Lambda}_s^{-1}\boldsymbol{U}^T$. Therefore,

$$(\star) \leq \left|\ln\left(1+\boldsymbol{U}(\boldsymbol{\Lambda}^{-1}-\boldsymbol{\Lambda}_s^{-1})\boldsymbol{U}^T\right) - \ln(1)\right| = \tag{4.19}$$

$$\left|\ln\left(1+\boldsymbol{U}(\boldsymbol{\Lambda}^{-1}-\boldsymbol{\Lambda}_s^{-1})\boldsymbol{U}^T\right)\right| \leq \tag{4.20}$$

$$\left|\ln\left(1+\alpha\cdot\sum_{i,j\in\mathcal{I}nv_{\mathcal{X}}(\pi)}(\boldsymbol{\Lambda}^{-1}-\boldsymbol{\Lambda}_s^{-1})_{ij}\right)\right|, \tag{4.21}$$

where $\mathcal{I}nv_{\mathcal{X}}(\pi)$ is the set of variables involved in (the hypothesis for) π, and the scalar α complies to $\alpha \geq \max_i \boldsymbol{U}(i)^2$. We recall that $\boldsymbol{X}(i)$ is uninvolved in the hypothesis $\iff \boldsymbol{U}(i) = 0$. Thus, we can easily consider additional hypotheses for π, and update this bound, by considering the involved variables, and the maximal value for α, among all the hypotheses. Further, by considering the involved variables and the maximal value of α, among the hypotheses of all candidates, this bound becomes independent of any specific action, and only a single expression needs to be calculated. Overall, we can conclude the following bound on the offset:

$$\boxed{\Delta(\mathcal{P}, \mathcal{P}_s) \leq \left|\ln\left(1+\alpha\cdot\sum_{i,j\in\mathcal{I}nv_{\mathcal{X}}(\Pi)}(\boldsymbol{\Lambda}^{-1}-\boldsymbol{\Lambda}_s^{-1})_{ij}\right)\right|.} \tag{4.22}$$

[2]For ease of writing, we will not explicitly mark matrix augmentation, whenever the context is clear

As we may notice, this symbolic bound depends on the initial belief of the original and simplified problems, yet not on their solution; it hence can be utilized before actually solving the problem. When calculating this bound, we considered only single-row collective Jacobians, but otherwise arbitrary. Guaranteed action consistency for the case of single-row Jacobians, which are also limited to a single non-zero entry, was previously shown by Indelman (2016).

4.3.2 Post-Solution Guarantees

For a more general scenario, when sparsifying involved variables, and with actions possibly having multi-row collective Jacobians, we can try to bound the loss by performing post-solution analysis, as discussed in Section 2.2. Unlike before, such guarantees are derived *after* solving the simplified problem (but before applying the selected action). As explained in Section 2.1.2, we can utilize the calculated (simplified) objective values, and domain-specific lower and upper bounds of the objective function ($\mathcal{LB}$,$\mathcal{UB}$, respectively), to yield offset bounds (2.11)-(2.12); from these offset bounds, we can then easily derive loss bounds (2.19).

Let us provide a couple of examples for the derivation of $\mathcal{LB}$ and $\mathcal{UB}$. Note that we will again assume that π corresponds to a single hypothesis. If this is not the case, we can simply calculate the bounds for every hypothesis, and take the most conservative bound among those.

Using Topological Properties

As our decision problem domain relies on beliefs, which, as we know, can be represented with factor graphs, we can potentially exploit their topological properties to derive the desired bounds.

For example, we can utilize conclusions from a recent work by Kitanov and Indelman (2019), which extends a previous work by Khosoussi et al. (2018). There, the following bounds on the information gain were proved, for when the corresponding factor graph contains only the agent's poses, and each pose consists of the position and the orientation of the agent (i.e., pose-SLAM):

$$\mathcal{LB}_{\text{top}}\left\{\tilde{V}(b,\pi)\right\} \doteq 3\cdot\ln t(b,\pi)+\mu+\mathrm{H}(b), \tag{4.23}$$

$$\mathcal{UB}_{\text{top}}\left\{\tilde{V}(b,\pi)\right\} \doteq \mathcal{LB}_{\text{top}}\left\{J(b,\pi)\right\}+\sum_{i=2}^{n}\ln(d_i+\Psi)-\ln\left|\tilde{\boldsymbol{L}}\right|, \tag{4.24}$$

where $t(b,\pi)$ stands for the number of spanning trees in the factor graph of the posterior belief (b after applying π); n marks the graph size; $\tilde{\boldsymbol{L}}$ is the reduced Laplacian matrix of the graph; and d_i's are the node degrees corresponding to $\tilde{\boldsymbol{L}}$. They also assume that the factors between the poses are described with a constant diagonal noise covariance; μ and Ψ are constants which depend on this noise model, and the posterior graph size (i.e., the length of the action sequence). In their demonstration, they show that when the ratio between the angular variance and the position variance is small, these bounds are empirically tight. This case can happen, for example, when a navigation agent is equipped with a compass, which reduces the angular noise. For a detailed derivation of these bounds please refer to Kitanov and Indelman (2019).

Using Determinant Inequalities

For different problem domains, it is possible to use various other objective bounds in a similar manner. For example, we present next additional bounds, which are relevant for Gaussian beliefs, and exploit known determinant inequalities.

For the lower bound, we can use *Minkowski determinant inequality*, which states that for positive semi-definite matrices $\boldsymbol{M}_1, \boldsymbol{M}_2 \in \mathbb{R}^{N\times N}$

$$|\boldsymbol{M}_1 + \boldsymbol{M}_2|^{\frac{1}{N}} \geq |\boldsymbol{M}_1|^{\frac{1}{N}} + |\boldsymbol{M}_2|^{\frac{1}{N}}, \tag{4.25}$$

$$\ln|\boldsymbol{M}_1 + \boldsymbol{M}_2| \geq N \cdot \ln\left(|\boldsymbol{M}_1|^{\frac{1}{N}} + |\boldsymbol{M}_2|^{\frac{1}{N}}\right). \tag{4.26}$$

Let us assign $\boldsymbol{M}_1 \doteq \boldsymbol{\Lambda}$ and $\boldsymbol{M}_2 \doteq {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta$, where $\boldsymbol{J}_\pi^\delta$ marks the collective Jacobian of (the hypothesis for) π, where n_π is the posterior state size. When ${\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta$ is not a full rank update, $\left|{\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta\right| = 0$, and we are left with

$$\ln\left|\boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta\right| \geq \ln|\boldsymbol{\Lambda}|. \tag{4.27}$$

For the upper bound, we can use *Hadamard inequality*, which states that for a positive semi-definite matrix $\boldsymbol{M} \in \mathbb{R}^{N\times N}$

$$|\boldsymbol{M}| \leq \prod_{i=1}^{N} (\boldsymbol{M})_{ii}. \tag{4.28}$$

Let us assign $\boldsymbol{M} \doteq \boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta$; then

$$\left|\boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta\right| \leq \prod_{i=1}^{n_\pi} (\boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta)_{ii}, \tag{4.29}$$

$$\ln\left|\boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta\right| \leq \sum_{i=1}^{n_\pi} \ln[(\boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta)_{ii}]. \tag{4.30}$$

Overall, by assigning this expressions in the objective function (3.33), we may conclude the following bounds:

$$\mathcal{LB}_{\det}\left\{\tilde{V}(b,\pi)\right\} \doteq \frac{1}{2}\left(\ln|\boldsymbol{\Lambda}| - N \cdot \ln(2\pi e)\right) + \mathrm{H}(b), \tag{4.31}$$

$$\mathcal{UB}_{\det}\left\{\tilde{V}(b,\pi)\right\} \doteq \frac{1}{2}\left(\sum_{i=1}^{n_\pi} \ln[(\boldsymbol{\Lambda} + {\boldsymbol{J}_\pi^\delta}^T \boldsymbol{J}_\pi^\delta)_{ii}] - n_\pi \cdot \ln(2\pi e)\right) + \mathrm{H}(b), \tag{4.32}$$

where $\boldsymbol{\Lambda}$ is the information matrix of b, $\boldsymbol{J}_\pi^\delta$ is the collective Jacobian of π, and n_π is the posterior state size.

Unlike the topological bounds, these bounds are extremely general, as they make no assumptions on the state structure nor actions, besides the standard problem formulation. As expected, this advantage comes at the expense of tightness. Nonetheless, they may especially be useful when the matrix $\boldsymbol{\Lambda}$ is diagonally dominant.

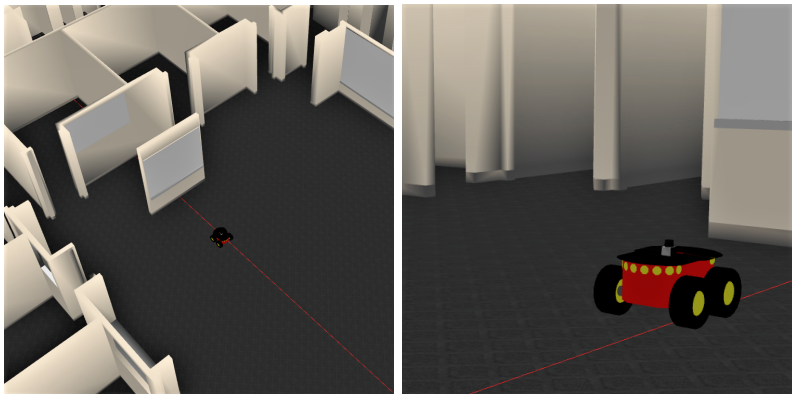


Figure 4.6: A Pioneer 3-AT robot in the simulated indoor environment. The robot is equipped with a lidar sensor, Hokuyo UST-10LX, as visible on top of it.

4.4 Experimental Evaluation

Next, we demonstrate the advantages of the solution approach, and present an in-depth analysis of it, in a highly realistic active-SLAM simulation.

4.4.1 Simulation Overview

In this simulation, a ground robot autonomously navigates through a sequence of goal points in an unknown indoor environment. We used the Gazebo simulation engine (Koenig and Howard, 2004) to simulate the environment and the robot (Pioneer 3-AT), which is equipped with a lidar sensor (Hokuyo UST-10LX), providing point-cloud observations; these components can be seen in Fig. 4.6. A grid-based map of the environment is incrementally approximated by the robot from the scans during the navigation. Robot Operating System (ROS) is used to run and coordinate the system components – state inference, decision making, sensing, etc.

The robot's state $\mathcal{X}_k \doteq \{x_0, \ldots, x_k\}$ consists of discretized poses kept along its entire executed trajectory. Its belief over this trajectory is represented as a factor graph, and maintained using the GTSAM C++ library (Dellaert, 2012). Each pose x_k consists of three variables, representing the 2D position and the orientation (in a global coordinate frame). The robot's initial location is provided to it as a prior factor over x_0 (with white Gaussian noise). After passing a set distance, a new pose is added to the state, and a lidar scan (in a range of approximately 30 meters) is taken. A motion constraint (pose transition factor), with white Gaussian noise, matching the real hardware's specs, is added between the new pose and the previous one. The observed point-cloud is then matched to scans taken from key past poses using ICP matching (Besl and McKay, 1992). If a match is found, a Gaussian loop closure constraint (factor) is added between these poses. After adding the new motion and observation constraints, the agent incrementally updates its belief and state estimate; we refer to this belief update process as a "state inference session". It is important to explicitly mention that despite examining a 2D navigation scenario, and relying on the "pose-SLAM" paradigm, our suggested approach does not impose any restrictions on the pose size nor on the state structure.

To initiate the simulation, and build a rudimentary map, the robot is set to follow a short predefined path, given as a sequence of way-points. To follow a desired path, the robot uses a "pure pursuit" controller Coulter (1992), which yields an appropriate sequence of control actions, based on the robot's estimated position (the mean of the belief).

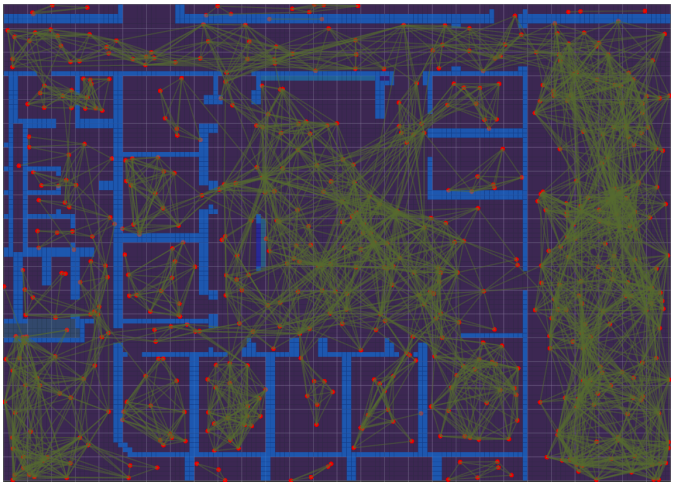


Figure 4.7: The entire indoor environment from a top view. Walls are colored in light blue. The PRM graph, from which trajectories are built, is colored in red and green. Each square on the map represents a 1m×1m square in reality.

4.4.2 Planning

After completing the initial predefined path, the robot is tasked to *safely* go through a a sequence of goal points in the environment; the robot should autonomously plan its path.

Based on its latest belief b (with a matching factor graph FG), the robot performs a "planning session", towards the first goal. It begins by generating a set Π of twenty collision-free candidate paths (sequences of way-points) from its current estimated location to the goal: it starts by executing the Probabilistic RoadMap (PRM) algorithm (Kavraki et al., 1996) to sample the map, and then executes the K-diverse-paths algorithm (Voss et al., 2015) on the sampled PRM graph, to return a set of topologically diverse paths. The full indoor map is unknown to the robot, and it is incrementally approximated by it using the scans during the navigation. We do, however, rely on the full and exact map to produce collison-free candidate trajectories. This usage of the map is irrelevant to the demonstration, since, in our formulation, we consider the candidate actions are given. The complete indoor map is shown in Fig. 4.7, with the sampled PRM graph on it.

We wish to select the safest (i.e., most "informative") path to the goal. To evaluate and compare the "safety" of taking each path, we use the previously-defined belief-based information-theoretic objective function $\bar{V}$ (3.33), which conveys the expected posterior belief entropy (uncertainty over the trajectory). As explained, to evaluate this function, we shall predict the future belief development for each candidate, if we choose to follow it. Surely, when executing a path which passes through a previously explored area, certain loop closures would be added to the belief, reducing its uncertainty; hence, taking different paths might lead to different loop closures, different posterior entropy, and different objective values. For tractability, we evaluate each candidate by predicting only a single future hypothesis, as explained in Sec. 3.3.2. Hence, each candidate path $\pi \in \Pi$ is pre-matched with a single factor graph FG^{δ}_{π}, containing the factors and poses to be added to the prior belief graph FG, after following the path.

For each candidate, loop closure factors are predicted between future and past poses, simply if their estimated location is within a close range (i.e., where we expect to add them when following this path); thus, we do not need to predict the raw laser scans, but only which poses would be

connected with a factor. The observed value, i.e., the transformation between these poses, is determined according to the ML observation assumption, i.e., the transformation between the *estimated* poses. After updating the initial belief according to each of the candidate hypotheses, the value of each candidate is computed, and the optimal path is selected (marking the end of this planning session), and then followed. After reaching the goal at the end of the selected path, the robot conducts a new planning session, towards its next goal, until all goals are reached. Meaning, the robot conducts a series of planning sessions, which are separated by sequences of state inference sessions.

4.4.3 Utilizing Belief Sparsification

To evaluate our method, in each planning session, we solved three decision problems, with each problem relying on another version of the initial belief. The robot's original initial belief accounts for the trajectory of poses executed up to that point (the entire inferred state). The other two versions are generated by sparsifying the original belief using Algorithm 3 – one considering partial sparsification, and one considering full sparsification. Overall, in each session, the three configurations of the decision problem are as follows:

1. $\mathcal{P} = \left(b, \Pi, \tilde{V}\right)$ – the original decision problem;
2. $\mathcal{P}_{involved} = \left(b_{involved}, \Pi, \tilde{V}\right)$ – after sparsification of the uninvolved variables, which is guaranteed to yield an action consistent problem. We remind again that uninvolved variables correspond to columns of zeros in the collective Jacobians.
3. $\mathcal{P}_{diagonal} = \left(b_{diagonal}, \Pi, \tilde{V}\right)$ – after sparsification of all variables, leading to a diagonal information matrix, but not necessarily action consistent.

For each configuration, we measured the objective function calculation time per candidate (which includes the proper belief update), along with the one-time calculation of the sparsification itself for the latter two. For a fair comparison, we chose to the detach the timing of the belief update and sparsification processes from any framework-specific implementation overhead. As explained in Section 3.2, since the system is Gaussian, the belief update process can actually be seen as updating the square root information matrix, using new Jacobian rows, representing new factors. Thus, from our GTSAM implementation, at each planning session, we extracted the square root matrix $\boldsymbol{R}$ of the initial belief, and each candidate's "collective Jacobian" matrix J_{π}^{δ}. Then, using Algorithm 3, we created the two sparsified versions of the prior matrix, $\boldsymbol{R}_{involved}$ and $\boldsymbol{R}_{diagonal}$, as detailed before. For each version of the prior square root matrix, and for each candidate, we performed (and timed) an incremental update of the matrix, using the standard MATLAB implementation of "`qr` update" (in agreement with the iSAM algorithm (Kaess et al., 2008))[3]; we could then easily extract the determinant of these triangular matrices, and calculate the objective values. Throughout the process, the variables (poses) maintained the order by which they were added to the state, and were not explicitly reordered.

On the whole, in each planning session, we measure the total decision making time for each of the three configurations. At the end of each session, we applied the action selected by configuration 1. Of course, in a real application we would only solve the problem using a single configuration; here we present a comparison of the results for different configurations. We also did not invest in smart selection of variables for sparsification, as even full sparsification achieved very accurate results.

[3]Since the calculated posterior matrices are discarded after each planning session, we did not care here to maintain their sparsity (by performing the update via partial re-factorization).

Table 4.1: Numerical summary for all sessions. "Uninvolved var. ratio" represents the percentage of uninvolved variables in the prior state. "Run-time" represents the reduction in decision making time in the specified configuration, in comparison to the original problem. "NNZ" represents the reduction in the number of non-zero entries in the prior square root matrix, after using the sparsification. "Sparsification time" represents the cost of this one-time calculation, out of the entire problem run-time.

Session	$\lvert\mathcal{X}\rvert$	$\mathcal{P}_{involved}$				$\mathcal{P}_{diagonal}$		
		Uninvolved var. ratio	Run-time	Sparsification time	Non zeros	Run-time	Sparsification time	NNZ
1	567	46%	-23%	3%	-76%	-55%	1%	-97%
2	762	74%	-34%	4%	-77%	-67%	1%	-98%
3	1182	60%	-66%	1%	-83%	-85%	1%	-99%
4	1269	69%	-70%	2%	-86%	-86%	2%	-99%
5	1341	65%	-67%	2%	-84%	-82%	2%	-99%
6	1392	44%	-52%	<1%	-61%	-80%	<1%	-99%

4.4.4 Results

In this section we present and analyze the results from a sequence of six planning sessions. Each of the Figs. 4.8-4.12 showcase a summary of one of the planning sessions; these summaries include several components:

(a) A screenshot of the scenario, which includes: the estimated map (blue occupancy grid); the current estimated position (yellow arrow-head) and goal (yellow circle); the trajectory taken up to that time-step (thin green line); the candidate trajectories from the current position to the goal (thick lines in various colors); and the selected trajectory (highlighted in bright green).

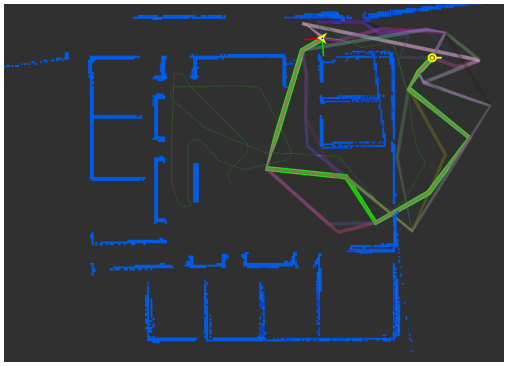
(b) A comparison of the objective function values of the candidate paths, considering each of the versions of the initial belief: $\mathcal{P}$ with the original belief in red; $\mathcal{P}_{involved}$ after sparsification of the uninvolved variables in blue; and $\mathcal{P}_{diagonal}$ after sparsification of all the variables in green. For scale, the values are presented in relation to the prior differential entropy, before applying any action. This prior value is not affected by the sparsification, and is the same for the three configurations (according to (4.11)).

(c) A comparison of the the solution time for the three decision problems: $\mathcal{P}$ in red, $\mathcal{P}_{involved}$ in blue, and $\mathcal{P}_{diagonal}$ in green. The highlighted parts of the blue and green bars mark the cost of the sparsification calculation out of the total solution time.

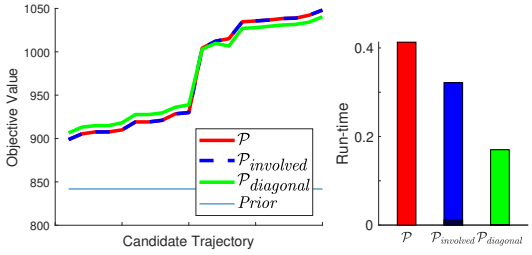
(d) A comparison of the three versions of the upper triangular square root matrix $\boldsymbol{R}$, $\boldsymbol{R}_{involved}$, and $\boldsymbol{R}_{diagonal}$. The figures showcase the NZ entries in each matrix, i.e., their sparsity pattern.

(e) The sparsity pattern of the collective Jacobians of the examined hypotheses. Again, uninvolved variables (in each hypothesis) are identified by columns of zeros in the Jacobian.

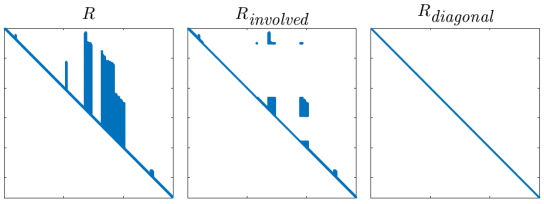
For the first and last sessions we provide an in-depth inspection, including all the components. Since the structure of the belief and Jacobians in all the sessions is similar, for the intermediate sessions we only present a summarized version, with components (a)-(c). The square root matrix and its approximations, given previously in Fig. 4.4, are extracted from the third session. Additionally, the numerical data shown in the figures is summarized in Table 4.1. Further data regarding the loss is later given in Table 4.2.



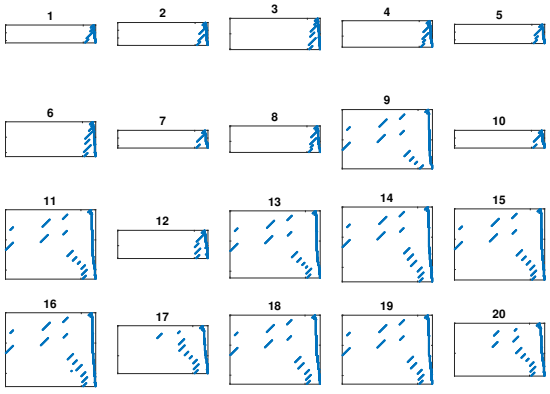
(a) The scenario: estimated map (blue occupancy grid); the current estimated position (yellow arrow-head) and goal (yellow circle); the executed trajectory (thin green line); the candidate trajectories from the current position to the goal (thick lines in various colors); and the selected trajectory (highlighted in bright green).



(b) Objective value comparison. **(c)** Run-time.

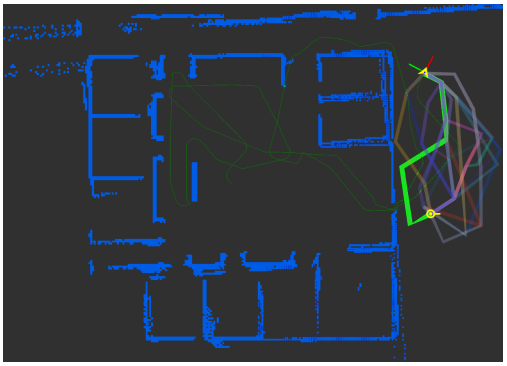


(d) Original prior information root matrix and its sparse approximations.

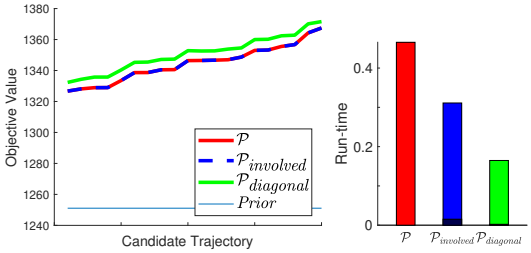


(e) Collective Jacobians of the candidate trajectories.

Figure 4.8: Results of planning session #1.

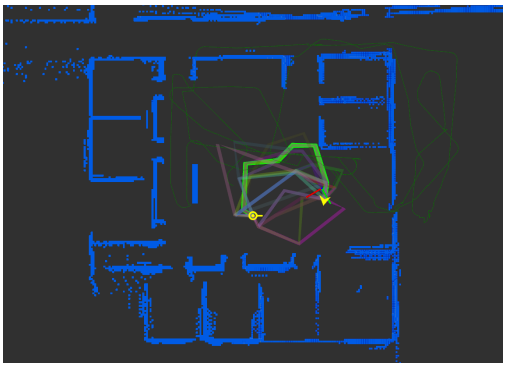


(a) The scenario.

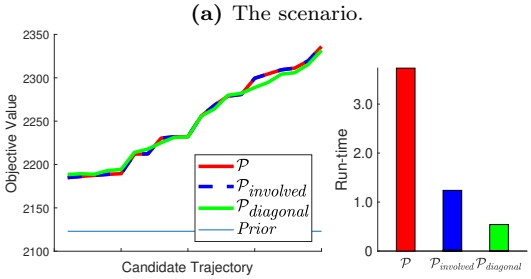


(b) Objective value comparison. **(c)** Run-time.

Figure 4.9: Results of planning session #2

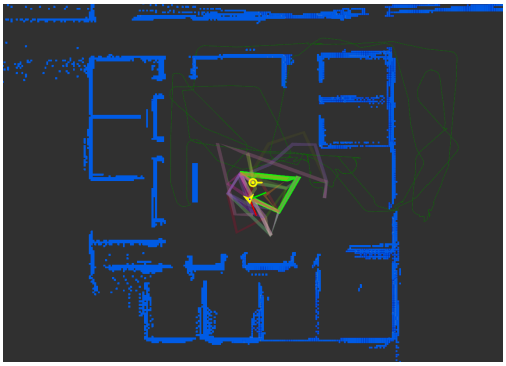


(a) The scenario.

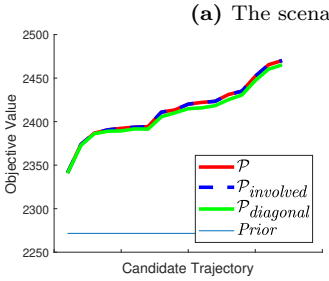


(b) Objective value comparison. **(c)** Run-time.

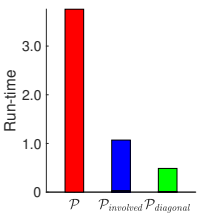
Figure 4.10: Results of planning session #3



(a) The scenario.

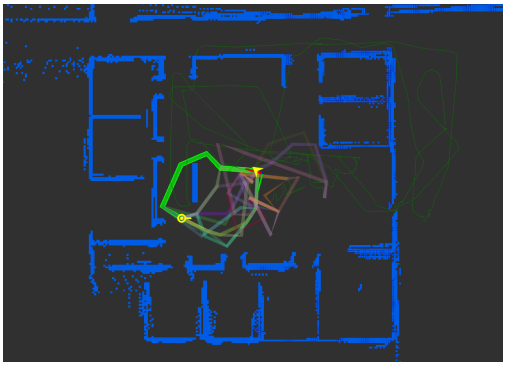


(b) Objective value comparison.

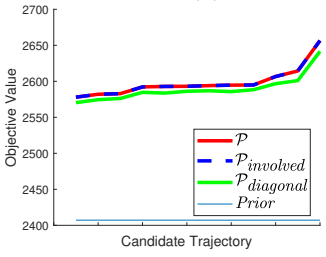


(c) Run-time.

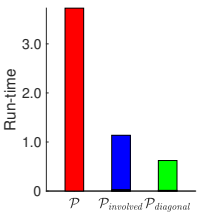
Figure 4.11: Results of planning session #4



(a) The scenario.

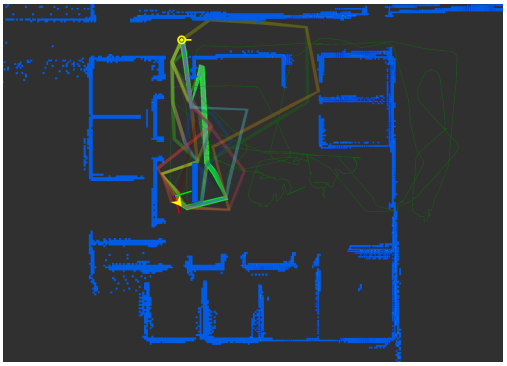


(b) Objective value comparison.

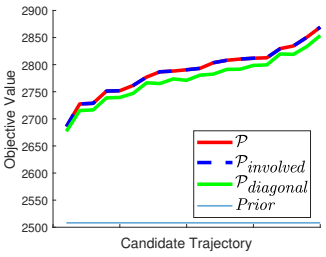


(c) Run-time.

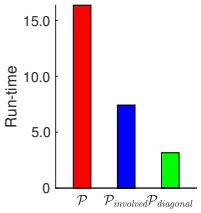
Figure 4.12: Results of planning session #5



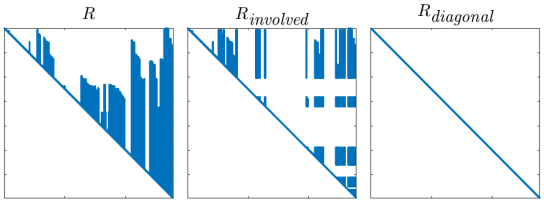
(a) The scenario: estimated map (blue occupancy grid); the current estimated position (yellow arrow-head) and goal (yellow circle); the executed trajectory (thin green line); the candidate trajectories from the current position to the goal (thick lines in various colors); and the selected trajectory (highlighted in bright green).



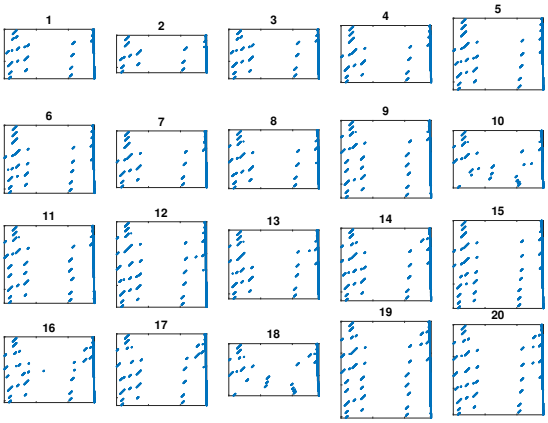
(b) Objective value comparison.



(c) Run-time.



(d) Original prior information root matrix and its sparse approximations.



(e) Collective Jacobians of the candidate trajectories.

Figure 4.13: Results of planning session #6.

Efficiency

As expected, the sparsification leads to a significant reduction in decision making time. The simplified problem $\mathcal{P}_{diagonal}$ consistently achieves the best performance, followed by $\mathcal{P}_{involved}$, while both are vastly more efficient than the original problem $\mathcal{P}$. Surely, a higher degree of sparsification ($\mathcal{S}$ containing more variables) leads to a greater improvement in computation time. As discussed in Section 4.1, full sparsification of the square root matrix has a particularly low cost – we only need to extract its diagonal. From Table 4.1 and the run-time comparison bar diagrams, it is clear that the cost of partial sparsification is also minor in relation to the entire decision making; in some of the diagrams, the highlighted section of the bar, which stands for the cost of the sparsification, is hardly visible. Also, since the sparsification cost does not depend on the number of candidate actions, the larger the set of actions is, the less significant the sparsifcation should become.

We see a correlation between the ratio of uninvolved variables and the reduction in run-time with $\mathcal{P}_{involved}$. Variables (poses) corresponding to the executed trajectory become involved when a loop closure factor is created between them and variables in the predicted trajectory. Hence, the ratio of uninvolved variables represents the overlap of the predicted trajectories with the previously executed trajectory. In session one, the executed trajectory is short, resulting in a relatively small state size, and sparse square root matrix, since not many loop closures were formed. As the sessions progress, the prior matrix becomes larger and denser, due to new loop closures, as apparent in session six.

In principle, we also notice a correlation between the state size and relative improvement in performance, for both sparsification configurations. As we may recall, updating the square root matrix (belief factorization), has, at worst, cubical complexity in relation to the number of affected variables. An update to a variable from the past, which appear at the beginning of the state (e.g., a loop closure) may force us to recalculate the entire factorization, baring this maximal computational cost. Sparsification of variables reduces the number of elements to update, and thus should be more beneficial when handling larger and denser beliefs.

Accuracy

Alongside the undeniable improvement in efficiency, we can also examine the "quality" of the selected candidates. According to Theorem 1, not only $\mathcal{P}$ and $\mathcal{P}_{involved}$ are action consistent, but they produce exactly the same objective values. Hence, solving $\mathcal{P}_{involved}$ always leads to the optimal action selection, and induces no loss. $\mathcal{P}_{diagonal}$ is not necessarily action consistent with the original problem, and therefore maintaining the same action selection is not guaranteed; however, it is evident from Figs. 4.8-4.13 that even when sparsifying all the variables using our algorithm, the quality of solution is maintained. Not only does the difference (offset) between the graphs of $\mathcal{P}$ and $\mathcal{P}_{diagonal}$ is slim, but they also maintain a very similar trend, leading practically to the same action selection, and no loss. This is also evident by examining the Pearson rank correlation coefficient ρ (which we mentioned in Section. 2.3) between the solutions of the original and simplified decision problems. A value of $\rho = 1$ represents perfect correlation of the candidate rankings (i.e., action consistency), and $\rho = -1$ represents exactly opposite rankings. Clearly, the calculated coefficients, presented in Table 4.2, indicate that $\mathcal{P}_{diagonal}$ indeed resulted in an action consistent solution (or very close to it). We emphasize again, that regardless of the configuration, the state inference sessions remain unchanged, as the selected action is always applied on the original belief.

Table 4.2: The actual loss induced by the two simplified configurations, alongside bounds on the loss (for the diagonal configuration), for different noise models. The specified ratio for each bound represents the ratio between the angular variance and the position variance. No bound is calculated for the other configuration, since it is guaranteed to induce no loss. The loss and its bounds are brought as a percentage of the maximal approximated value in that session. Also shown is Pearson rank correlation coefficient ρ.

Session	$\rho(\mathcal{P}, \mathcal{P}_{involved})$	$\rho(\mathcal{P}, \mathcal{P}_{diagonal})$	$loss(\mathcal{P}, \mathcal{P}_{involved})$	$loss(\mathcal{P}, \mathcal{P}_{diagonal})$	**Bound** "0.01:1"	**Bound** "0.25:1"	**Bound** "0.85:1"
1	1	0.99	0%	0%	2%	16%	46%
2	1	1	0%	0%	2%	16%	47%
3	1	1	0%	0%	1%	13%	39%
4	1	0.99	0%	0%	1%	15%	43%
5	1	1	0%	0%	1%	16%	43%
6	1	0.99	0%	0%	1%	15%	41%

Guarantees

Throughout the experiment, after each (simplified) planning session, it was possible to guarantee the quality of solution for $\mathcal{P}_{diagonal}$, by bounding $loss(\mathcal{P}, \mathcal{P}_{diagonal})$ in post-solution analysis (before applying the selected action). Obviously, no bound should be calculated for $\mathcal{P}_{involved}$, since the loss was guaranteed to be zero in our pre-solution "offline" analysis. As explained in Section 2.2, we can derive a loss bound using the solution of the simplified problem (which is available), and some domain-specific bounds/limits for the objective function. Here, we used the topological bounds (4.23)-(4.24), which we covered in Section 4.3.2, and assigned them in (2.19), in order to derive the appropriate guarantees.

The tightness of these topological bounds, which affects the tightness of the loss bound, depends on the ratio between the angular variance, and the position variance, with which we model the noise in factors between poses; the smaller the angular noise is, in relation to the latter, the tighter the bounds are (as analyzed by Khosoussi et al. (2018) and by Kitanov and Indelman (2019)). Hence, we calculated the loss bound assuming different noise models (different such ratios), and examined their effects. Such a change to the noise model has a minor effect on the objective evaluation, since it does not change the sparsity pattern of the matrices; thus, we only present the effect on the inferred loss bound, and not on the entire planning process. The bounds, which were calculated assuming different noise ratios, are given in Table 4.2. The loss and its bounds are brought as a percentage of the maximal approximated objective function value in that session, to allow a correct comparison. In the scenario showcased before, the angular variance to position variance ratio was 0.25:1.

Indeed, changing the noise model has a significant influence on the tightness of the loss bounds. A ratio of 0.01:1 yields a very tight bound. It is not far-fetched that the angular variance would be this low in a navigation scenario, for example, by having a compass, as mentioned before. Raising this ratio results in more conservative bounds, especially in comparison to the exact loss, which is zero. Yet, they can still be used to guarantee that the solution stays in an acceptable range. Developing tighter bounding methods for the objective function shall help making these guarantees less conservative.

To clarify, this discussion, alongside any assumptions on the noise or state structure, is only brought in order to examine our ability to provide guarantees, using this specific topological approach. It is not essential in any way to the applicability of this belief sparsification.

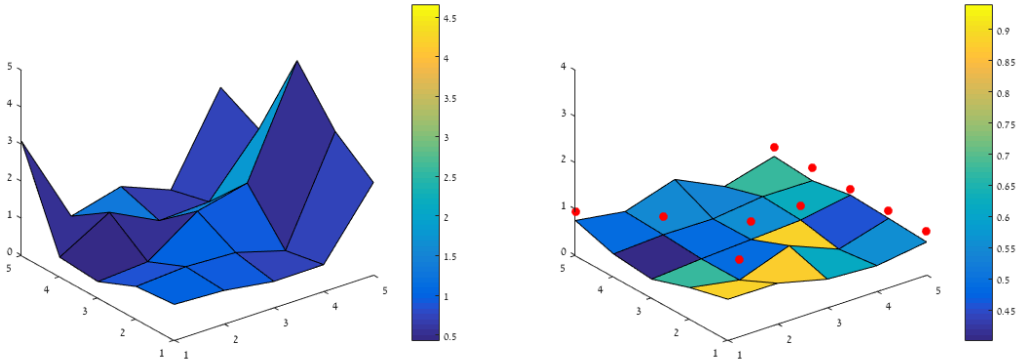


Figure 4.14: Uncertainty levels in a temperature grid – before (left) and after (right) sensor placement. Higher values indicated higher uncertainty levels in that junction. We wish to find the optimal sensor placement on the grid junctions.

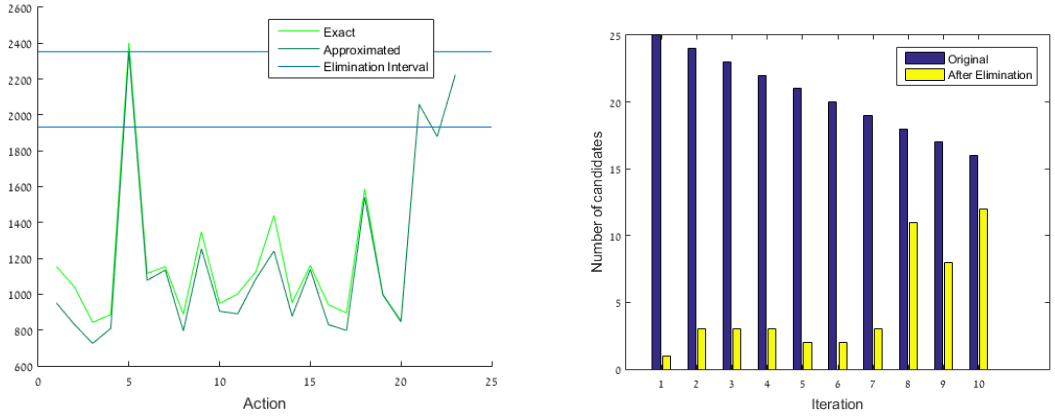
4.5 Experimental Demonstration: Action Elimination

As we recall, our belief sparsification algorithm (Algorithm 3) relies on a selection of a set $\mathcal{S}$ of variables for sparsification. By selecting which and how many variables to sparsify, one can control the "scale" of the simplification. Further, as explained in Section 4.3.1, for an initial belief approximation generated using Algorithm 3, and considering candidate actions which reflect rank-1 updates, the simplification offset can be bounded with a symbolic expression. This bound depends solely on the initial belief and a bound on the candidates' Jacobian entries, and can be derived ahead of planning. These properties makes the sparsification algorithm an appropriate match to the sequential action elimination algorithm (Algorithm 1), provided in Section 2.2.3. We would like to practically demonstrate this concept here.

4.5.1 Sensor Placement Simulation

In this simulation, we demonstrate the application of belief sparsification on another common example of sequential decision making under uncertainty – the sensor placement problem – which can also be modeled as decision making in the belief space. Here, we wish to place 10 sensors on the junctions of a 5×5 grid, in order to measure its temperature in the most informative way. This stochastic grid state is represented as a Gaussian belief b over the state vector $\boldsymbol{X} \in \mathbb{R}^{25}$, which represents the current temperature measurement in the 25 grid junctions. The problem is visualized in Fig. 4.14.

The scenario is initialized with a random belief over the grid, including randomized correlations between the junctions; such a belief is hence expressed with a dense prior information matrix $\boldsymbol{\Lambda}$. At each of the 10 planning sessions, we wish to greedily choose a junction from those available, and place a sensor on it (meaning, measure the temperature from that position). Since the measurements are scalar, each candidate (placement) can indeed be described with a "rank-1" update to the belief, i.e., a single-row collective Jacobian; such a Jacobian is a binary (0/1) row-vector, indicating the location of the sensor. The objective is, as before, to select the candidate which would minimize the belief's entropy (3.33).



(a) **(b)**

Figure 4.15: (a) The elimination process from the third planning session. The approximated values and the (unknown) exact values are shown in turquoise and green, respectively. Actions for which the approximated value is not within the elimination interval can be eliminated. (b) The number of candidate actions to examine in each planning session, before and after the elimination process.

At each planning session, we performed a single candidate elimination iteration, according to the previously described approach: we started by performing a *full* sparsification of the belief's information matrix, according to Algorithm 3, which left us with a diagonal matrix. We then used it to approximated the values of all candidates (free junctions). Based on these values, and the offset bound for rank-1 updates (4.22), we could then infer the elimination threshold τ_1 (2.24).

Fig. 4.15a shows a single elimination process from the third planning session. The approximated values and the (unknown) exact values are shown in turquoise and green, respectively. The horizontal blue lines stand for the limits of the elimination interval (between the maximal approximated value and τ_1); actions for which the approximated value is not between these lines, can be eliminated. In this iteration, only 3 candidates passed the elimination. We could then solve the problem by calculating the exact values for only these candidates, or perform another elimination iteration, using a refined belief approximation (sparsification of less variables).

Fig. 4.15b shows the number of remaining candidates after the elimination, in comparison to the original number of candidates, in each planning session. In a few cases, all candidates besides one were eliminated, hence leaving us with the optimal candidate, without calculating any of the values exactly. Note that in the later sessions, the most informative sensors were already placed, and the remaining candidates yield similar values (i.e., no dominant actions); therefore, more candidates managed to pass the elimination.

Chapter 5

PIVOT: Predictive Incremental Variable Ordering Tactic

Evidently, using belief sparsification as a simplification method proved to be very effective in reducing the cost of planning in the belief space. However, one of the main weaknesses of that method is that the sparsified belief is discarded after every planning session; thus, whenever re-planning is due, we need to re-calculate the sparsification "from scratch". To address this issue, we explore next another simplification approach – one which can be updated incrementally.

5.1 PIVOT

Consider a prior belief $b(\boldsymbol{X})$, and a set of future hypotheses $\mathcal{H} \doteq \{H_1, \ldots, H_m\}$, as we defined before, in Chapter 4. As discussed, we wish to update b according to each of these hypotheses, i.e., calculate b_H, $\forall H \in \mathcal{H}$. Yet, we recall that to perform state inference, we are actually interested in updating the factorized representation of b, which depends on the initial variable order expressed in $\boldsymbol{X}$. This representation is graphically represented as the Bayes net BN in the general case, or as the square root information matrix $\boldsymbol{R}$ in the Gaussian case. We previously explained how these updates can be performed via incremental re-factorization – only considering the variables starting from the first variable involved in the update (the "affected variables"). Notably, the initial variable order determines the index of the first involved variable in each of the future updates, and, by such, which variables would be affected by each of them. We should consider that updating the square root factorization has, at worst, cubical complexity in relation to the number of affected variables (Hämmerlin and Hoffmann, 2012); hence, for computational efficiency concerns, we wish this number, in each update, to be minimal.

We thus suggest the following concept: optimize the initial variable order predictively – i.e., before and in preparation to performing these updates – in order to increase their efficiency. In practice, we wish to perform a precursory and standalone modification to the factorized belief representation, to convey reordering of the variables. This reordering (which may require partial re-factorization) surely comes at a cost, but, as to be seen, this small investment in the current time should be overall beneficial in the long run, by reducing the cost of future belief updates. We shall clarify that, performing this standalone reordering only makes sense when facing multiple "parallel" updates, as happens, e.g, in the context of planning. If we face only a single update, e.g., as occurs in regular state inference updates, then both the update and the reordering can been performed together during the same re-factorization; such re-factorization would, anyway,

affect the variables starting from the first involved/reordered variable.

The question remaining is how to actually determine the optimal variable order in b, considering multiple future updates for it. Next, to answer this question, we will examine several "predictive variable ordering tactics". Though, beforehand, to avoid confusion, we wish to clarify that the discussion to follow only comes to answer the question "what should be the new variable (elimination) order?". The question "how to modify the existing square root matrix $\boldsymbol{R}$ (or Bayes net BN), after the new order is determined?" is extensively discussed in Chapter 6, and one should refer to it for full details.

5.1.1 Pushing Forwards Involved Variables

We note that for each update, the set of affected variables, those which should be re-eliminated when performing the update, may include also uninvolved variables, which appear after the first involved variable (according to the initial variable order); such variables are only (unnecessarily) affected due to their index in the state. We wish to determine our variable ordering tactic such that we avoid such scenarios. Beforehand, let us clarify our terminology, as used throughout this chapter: pushing a variable forwards means increasing its index in the state vector, bringing it closer to the vector's end; accordingly, pushing a variable backwards means decreasing its index, bringing it closer to the vector's start.

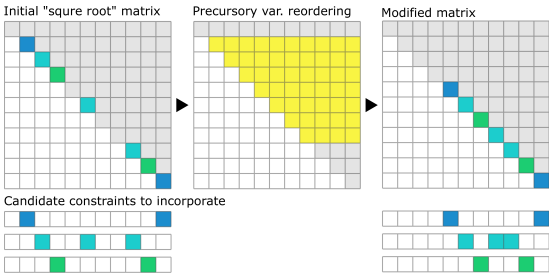
Thus, according to our ordering tactic, we suggest to push backwards all the uninvolved variables $\neg\mathcal{I}nv_{\mathcal{X}}(\mathcal{H})$, to appear before $\mathcal{I}nv_{\mathcal{X}}(\mathcal{H})$, or equally, **push the involved variables forwards**; this verifies that variables that are uninvolved in all the hypotheses, are never affected while planning. This approach, which we refer to as *PIVOT: Predictive Incremental Variable Ordering Tactic*, can be represented with the permutation

$$\texttt{PIVOT}_1(\boldsymbol{X}) \doteq \begin{bmatrix} \neg\mathcal{I}nv_{\boldsymbol{X}}(\mathcal{H}) \\ \mathcal{I}nv_{\boldsymbol{X}}(\mathcal{H}) \end{bmatrix}, \tag{5.1}$$

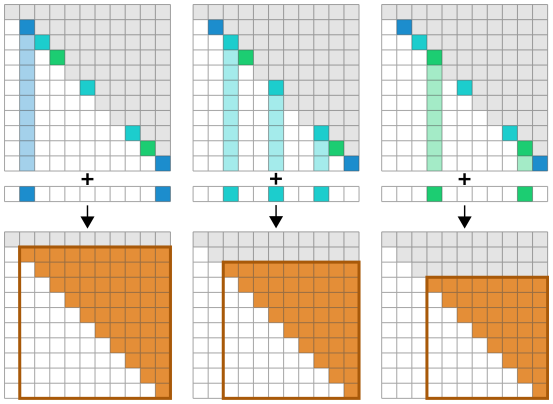
which separates the variables according to their *class*: involved or uninvolved. Optimally, for each hypothesis, we would like to update only the involved variables, and make all the uninvolved variables unaffected. However, since we wish to perform the reordering only *once* (per planning session), some of the affected variables, for each hypothesis, might still not actually be involved. Nonetheless, such reordering maximizes the number of unaffected variables, and, by such, the overlapping sub-matrix of the prior square root matrix/sub-graph of prior Bayes net, which can best be reused when calculating all posteriors. This concept is demonstrated in Fig. 5.1.

We note that the separation of variables **maintains the relative order** of variables in each category. I.e., if a variable x initially came before a variable y, and they both are in the same class, then even after the reordering, their relative order is kept. This means that involved variables may only be pushed forwards (compared to their original index), and the number of affected variables in each planning hypothesis may only decrease.

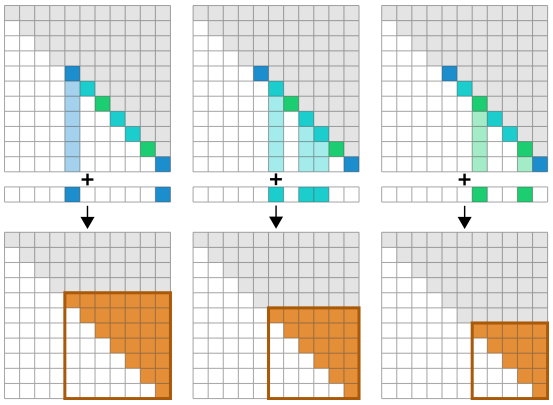
Nonetheless, while maintaining the relative variable order in the "involved" class presumably makes sure that the performance of each update it bettered, this property is not always desirable when considering the overall planning performance point of view. For example, if a variable is only involved in one hypothesis, it might not be sensible to keep it ahead of another variable which is involved in all actions, as it would cause the first variable to unnecessarily be affected in all updates. Instead, we (might) want the order among the affected variables to convey their "involvement level", i.e., the number of hypotheses in which a variable is involved. In this case, pushing "more forwards" variables which are "more involved", may increase the number of affected variables in some of the updates, but the total number of affected variables, considering all the updates, would be minimized.



(a) On the left are the sparsity pattern of the square root matrix $\boldsymbol{R}$ of a belief b, and the Jacobians of three candidate updates; these may generally contain multiple rows, and also reflect addition of variables. Each state variable corresponds to a row/column in $\boldsymbol{R}$, according to the variable order. The colored cells in each update's Jacobian matrix signifies non-zero columns, which in turn represent the involved variables in the update. According to $\texttt{PIVOT}_1$, we should reorder the state variables, and push forwards the involved variables, towards the end of the state. The modified matrix, after variable reordering appears on the right; of course, we must reorder the new Jacobians appropriately, such that the order of their columns matches the new variable order.



(b) Updating the original matrix according to each of the candidates. The orange blocks in each posterior matrix represent the affected (i.e., recalculated) entries; this block is determined according to the index of the first involved variable (column).



(c) By applying a precursory reordering step, and using the modified matrix, we are able to reduce the size of affected block in each update.

Figure 5.1: A conceptual demonstration of $\texttt{PIVOT}_1$. Please note that we do not assume dense matrices; gray or orange-colored cells only indicate entries which *may* be non-zero.

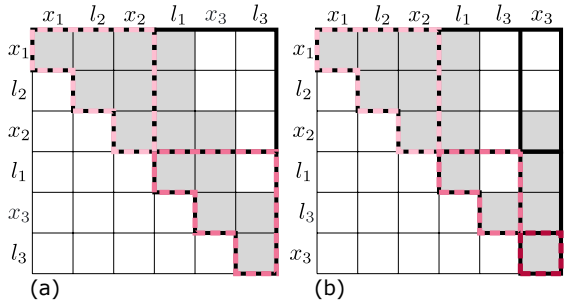


Figure 5.2: The block structure of the modified prior square root matrix $\boldsymbol{R}$, after applying $\texttt{PIVOT}_1$ (a) and $\texttt{PIVOT}_2$ (b), in the planning scenario presented Fig. 4.5. $\texttt{PIVOT}_1$ places all involved variables (l_1, x_3, l_3) in the same class; $\texttt{PIVOT}_2$ separates the involved variables into two classes, as x_3 is involved in both candidates, while l_1 and l_3 are involved in one candidate, each.

We thus suggest a generalization of $\texttt{PIVOT}_1$, in which we subdivide the involved variables into $c \in \mathbb{N}$ (sub-)classes, according to their involvement level:

$$\texttt{PIVOT}_c(\boldsymbol{X}) \doteq \begin{bmatrix} \text{class}^0_{\boldsymbol{X}} \\ \vdots \\ \text{class}^c_{\boldsymbol{X}} \end{bmatrix}, \tag{5.2}$$

where

$$\text{class}^i_{\mathcal{X}} \doteq \left\{ x \in \mathcal{X} \mid (i-1) \cdot \frac{M}{c} < \texttt{level}(x) \leq i \cdot \frac{M}{c} \right\} \tag{5.3}$$

$$M \doteq \max_{x \in \mathcal{X}} \texttt{level}(x) \tag{5.4}$$

$$\texttt{level}(x) \doteq \sum_{H \in \mathcal{H}} 1\left(x \in \mathcal{I}nv_{\mathcal{X}}(H)\right) \tag{5.5}$$

For example, when $c = 1$, the class separation is based on a binary indicator of involvement – this matches the definition of $\texttt{PIVOT}_1$ from before, where all the involved variables are in the same class. When $c = 2$, we subdivide the involved variables into two classes – those which are involved in at most $\frac{M}{2}$ updates, and those who are involved in more than that amount (where M is the maximum number of updates in which any variable is involved). We can similarly continue to increase c for a finer division. We may also explicitly require $c = \max$, to treat every involvement level as a distinct class. For any choice of c, $\text{class}^0_{\mathcal{X}} \equiv \neg\mathcal{I}nv_{\mathcal{X}}(\mathcal{H})$ always contains the uninvolved variables. Algorithm 4 summarizes the steps for calculating $\texttt{PIVOT}_c$. Further, for any chosen c, we can break the modified prior square root matrix into a block structure, according to the variable class division, as demonstrated in Fig. 5.2.

5.1.2 Fill-Aware Ordering

As reviewed, pushing the involved variables forwards is an ordering tactic, intended to keep the cost of future belief updates (during planning) low; however, this tactic is not always optimal. Indeed, when handling dense matrices (beliefs), then the cost of re-factorization (updates) is determined solely by the number of variables in the affected blocks. Yet, in sparse systems, the cost of such updates also tightly depends on the density of these blocks.

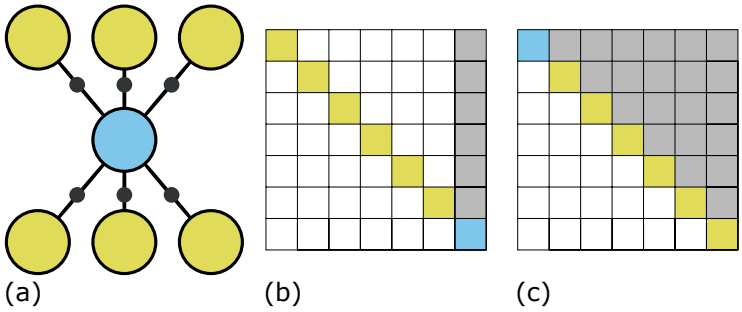


Figure 5.3: (a) A factor graph representing a small belief; the graph contains a "highly connected" variable (in blue). We can examine the sparsity pattern of the belief's square root matrix, which represents the structure of its Bayes net, after variable elimination: (b) shows the sparsity pattern, if the "highly connected" variable is eliminated last; (c) shows the pattern if the variable is eliminated first. Evidently, to reduce fill-in, "highly connected" variables should (typically) be placed forwards.

It is theoretically possible that applying the previous tactic, and reducing the size of an affected block, would actually make the cost of re-factorization higher, if this operation introduced more fill-in in this block (i.e., made it denser). Of course, we refer here to the density of the affected block in the *prior* square root matrix; yet, this acts as the baseline for the fill-in in the re-calculated block in the posterior matrix, which can only increase after adding new constraints. In other words, a dense (block in the) prior would lead to a dense (block in the) posterior, which conveys high re-factorization cost. This fill-in would also affect the cost of performing back-substitution, which is needed in order to update the MAP estimate. Further, the fill-in in the prior also conveys the cost of the reordering process itself, which also typically requires re-factorization. Thus, we would like to optimize the variable order such that we find balance between reducing the size of the affected blocks, and keeping the system density low. Taking this aspect into account, we shall reconsider the rules for class separation.

As a rule of thumb, to reduce fill-in, "highly connected" variables should be placed forward, after the variables it is connected to; placing it behind these variables (i.e., eliminating it first, which is equivalent to its marginalization) would add new factors involving all the connected variables, and lead to a dense block. This property (which is demonstrated in Fig. 5.3) is the key to "minimum degree" fill-reducing ordering strategies.

Thus, considering a certain update, having a "highly connected" (uninvolved) variable pushed back, above its first involved (i.e., outside of the affected block), can make the affected block dense, and actually decrease the re-factorization efficiency! To avoid such scenarios, we want to identify these "highly connected" variables, and push them further forward in the variable order, by bumping them to higher classes. We thus consider the following adjustment to the original involvement-level-based class separation rule: if a variable has more factors connecting it to (variables in) higher classes, than factors connecting it to (variables in) its own class or lower ones, then forcefully bump it to a higher class. We mark this fill-optimized variable reclassification as $*$class.

As another treatment for the fill-in issue, we may combine $\texttt{PIVOT}_c$ with a fill-reducing order, such as `CCOLAMD`. This order is the constrained version of the famous `COLAMD`, which can be used to enforce the relative order among groups of variables, by specifying a certain value $\in \mathbb{N}$ ("constraint") for every variable. Then, for example, all variables with value 1 will appear before variables with value 2, which will appear before variables with value 3, and so on. The internal ordering of variables with the same constraint will match the fill-reducing `COLAMD`. In our case,

Algorithm 4: `PIVOT`

Inputs:

An initial belief b (with state vector $\boldsymbol{X}$), and a set $\mathcal{H}$ of future hypotheses, as described in Sec. 5.1
c (chosen number of classes),
FILL_AWARE (boolean flag),
FORCE_INCREMENTAL (boolean flag)

Output:

A permuted state vector $\tilde{\boldsymbol{X}}$

`// calc variable involvement levels`
class,`level` $\leftarrow$ $zeros(1, n)$ `// init zero vectors`
forall $x \in \mathcal{X}$ **do**
  forall $H \in \mathcal{H}$ **do**
    if $x \in \mathcal{I}nv_{\mathcal{X}}(H)$ **then**
      `level`$(x)++$

`// divide involved vars into` c `classes`
if $c = 1$ **then**
  forall $x \in \mathcal{X}$ **do**
    class$(x) \leftarrow \max\{1,$ `level`$(x)\}$
else if $c = \max$ **then**
  class$(x) =$ `level`(x)
else
  $M \leftarrow \max_{x \in \mathcal{X}}\{$`level`$(x)\}$
  forall $x \in \mathcal{X}$ **do**
    class$(x) \leftarrow \min\left\{i \in [0, \ldots, c] \mid$ `level`$(x) \leq i \cdot \frac{M}{c}\right\}$
if *FILL_AWARE = false* **then**
  `// sort variables by class`
  $\tilde{\boldsymbol{X}} \leftarrow$ `sort`$(\boldsymbol{X},$ class$)$
else
  `// utilize fill-aware optimizations`
  if *FORCE_INCREMENTAL = true* **then**
    $j_{\text{first}} \leftarrow \min\{j \in [1, \ldots, n] \mid$ `level`$(x_j) > 0\}$ `// first involved var`
  else
    $j_{\text{first}} \leftarrow 1$ `// first state var`
  `// push forward highly connected vars`
  $*$class $\leftarrow$ class
  $\boldsymbol{\Lambda} \leftarrow \boldsymbol{J}^T\boldsymbol{J}$
  forall $j \in [j_{first}, \ldots, n]$ **do**
    `// count how many factors connect` x_j `to each class`
    connections $\leftarrow zeros(1, c)$
    forall $i \in [1, \ldots, j-1, j+1, \ldots, n]$ **do**
      if $\boldsymbol{\Lambda}(i, j) \neq 0$ **then**
        `// add a connection between` x_j `to the class of` x_i
        connections$(*\text{class}(x_i))++$
    current_class $\leftarrow$ class(x_j)
    while $\sum connections(1 : current\_class) < \sum connections(current\_class + 1 : c)$ **do**
      `// bump var to a higher class`
      current_class $++$
    $*$class$(x_j) \leftarrow$ current_class
  `// utilize a fill-reducing order`
  new_order $\leftarrow$ `CCOLAMD`$(\boldsymbol{J}, *\text{class})$
  if *FORCE_INCREMENTAL = true* **then**
    `// keep original order of vars preceding the first involved`
    new_order$(1 : j_{\text{first}} - 1) \leftarrow [1 : j_{\text{first}} - 1]$
  $\tilde{\boldsymbol{X}} \leftarrow \boldsymbol{X}$(new_order)

clearly, the variable constraints would match the fill-optimized variable classes, as defined before, to yield the fill-optimized PIVOT order:

$$\texttt{PIVOT}_c^*(\boldsymbol{X}) \doteq \texttt{CCOLAMD}(\boldsymbol{J}, \texttt{constraint}), \quad \text{where } \texttt{constraint}(x) \doteq i, \quad \text{such that } x \in *\text{class}_{\mathcal{X}}^{i} \tag{5.6}$$

The steps for calculating $\texttt{PIVOT}_c^*$ are (also) provided in Algorithm 4.

Note that using this tactic, the relative order of variables in each class is no longer maintained. We need to take into account that although considering more classes (larger c) shall lead to smaller affected blocks while planning, it also means adding more constraints to CCOLAMD, which can potentially lead to more fill-in. In contrast, choosing a small c may result in having numerous (involved) variables in each class; as the variables in each class can now be "shuffled" according to the fill-reducing order, this may counteract the minimization of the affected blocks. We thus expect the "optimal" selection of c to often be some middlepoint between 1 and max.

5.1.3 Incremental Reordering

Maintaining the relative order in the "uninvolved" class, as occurs in $\texttt{PIVOT}_c$, means that if the current variable order (i.e., $\boldsymbol{X}$) begins with uninvolved variables, they would keep their position, and not be reordered, i.e.,

$$\big[\texttt{PIVOT}_c(\boldsymbol{X})\big](1, \ldots, j-1) \equiv \boldsymbol{X}(1, \ldots, j-1), \tag{5.7}$$

where

$$j \doteq \min_{x \in \mathcal{X}} \{\texttt{index}(x, \boldsymbol{X}) \mid x \in \mathcal{I}nv_{\mathcal{X}}(\mathcal{H})\}, \tag{5.8}$$

marks the index of the first involved variable (in any of the updates). As later clarified in Chapter 6, this property means that the $\texttt{PIVOT}_c$ order can often be *applied* incrementally, with only partial modification of the belief.

This property (which is demonstrated in Fig. 5.4) is especially important for large systems, where performing "full" reordering may not be feasible in real time. However, it is not necessarily kept in the fill-aware variation of PIVOT, where variables in each class might be "shuffled" to a fill-reducing order. If we want to make sure the returned order can still be applied incrementally, we should consider certain adjustments to it. To begin with, we shall divide $\text{class}_{\mathcal{X}}^0$ (the uninvolved variables) into two sub-classes:

$$\text{class}_{\mathcal{X}}^0 = \text{class}_{\mathcal{X}<j}^0 \cup \text{class}_{\mathcal{X}>j}^0 \tag{5.9}$$

where

$$\text{class}_{\mathcal{X}<j}^0 \doteq \{x \in \text{class}_{\mathcal{X}}^0 \mid \texttt{index}(x, X) < j\} \tag{5.10}$$

contains all the uninvolved variables that already appear at the beginning of the state, before the first involved; and

$$\text{class}_{\mathcal{X}>j}^0 \doteq \{x \in \text{class}_{\mathcal{X}}^0 \mid \texttt{index}(x, X) > j\} \tag{5.11}$$

contains all the uninvolved variables that appear after first involved, and should be pushed back. This division is also demonstrated in Fig. 5.4. Then, to avoid affecting the order of the first j variables, our fill-aware optimizations should be adjusted as follows: first, the reclassification of "highly connected variables" should ignore the variables in $\text{class}_{\mathcal{X}<j}^0$; second, when applying CCOLAMD, the constraint value used for the variables in $\text{class}_{\mathcal{X}<j}^0$ should be set to -1, to force

them backwards; third, after the application of CCOLAMD, we should "cancel" the internal ordering of those variables, by forcing:

$$\left[\texttt{PIVOT}_c^*(\boldsymbol{X})\right](1,\ldots,j-1) = \boldsymbol{X}(1,\ldots,j-1) \tag{5.12}$$

These steps are explicitly formulated as part of Algorithm 4.

Note that, although such incremental application of $\texttt{PIVOT}_c^*$ is more efficient, it can lead to slightly inferior fill reduction in comparison to the standard counterpart. Still, the sub-optimal fill-in in the "uninvolved" rows does not affect the efficiency of the updates, as they involve variables with higher indices.

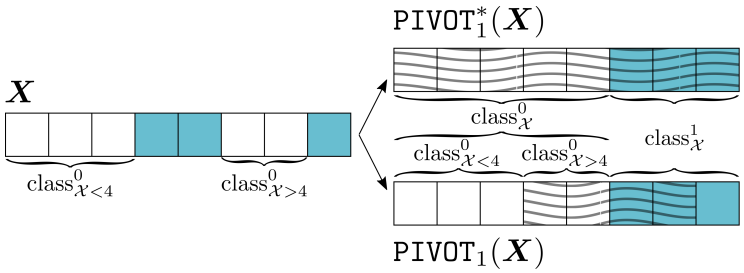


Figure 5.4: On the left – a visualization of a certain variable order, where white cells represent uninvolved variables, and blue cells represent involved variables (according to some hypotheses). On the right – two modified orders, returned from $\texttt{PIVOT}_1$ (bottom), and its fill-aware counterpart $\texttt{PIVOT}_1^*$ (top). We can see that $\texttt{PIVOT}_1$ does not change the order of uninvolved variables that appear at the beginning of the state (before the first involved variable at index 4); it can thus be applied incrementally. On the other hand, $\texttt{PIVOT}_1^*$ may cause reordering of all variables (marked with black overlay).

5.2 Utilizing PIVOT for Efficient Planning

Now, say we are interested solving the decision problem $\mathcal{P} = (b, \Pi, V)$, as described before in Section 3.3 (note that here, we do not necessarily consider the information-theoretic objective $\tilde{V}$). At this point in time, to calculate the candidate policies' values, we should perform multiple updates to the initial belief, propagating it according to the predicted hypotheses for the candidates. Suitably, the concept of PIVOT explains how to modify (simplify) a belief representation whenever facing multiple updates, in order to reduce their computational cost. Thus, let us bridge the gap, and explain how we can practically utilize PIVOT as a simplification method that allow to efficiently solve this decision problem.

5.2.1 True Classification

As explained, to determine the PIVOT order, we need access to the predicted updates, in order to determine the variable involvement levels and classification.

ML Observations

We recall that assuming "maximum likelihood (ML)" observations contributes to planning in two manners: (1) each candidate policy is represented with only a single hypothesis, and can be evaluated with a single update; (2) this hypothesis (along the entire planning horizon) can be inferred ahead of planning, without propagating the belief. Both of these properties are beneficial

for PIVOT, as they allow us to infer the "true" variable involvement levels in the long-horizon future updates, ahead of planning time; in other words, they allow us to "prepare" in advance for all the updates due in the planning process. Thus, in this case, we can determine a PIVOT order and apply it once *on the initial belief*, ahead of performing the updates (similarly to belief sparsification); this is demonstrated in Fig. 5.5a.

Note that, generally, we might still need to perform these predetermined updates "gradually" (step-by-step), in order to calculate the intermediate rewards. If we only measure the reward at the final time-step, then, for every hypothesis, we would calculate a single update, representing the entire hypothesis.

General Observations

When not relying on ML observations, each candidate is matched with multiple hypotheses. Further, for each candidate, and at each time-step, we have access only to the (multiple) immediately upcoming updates, taking the belief to the next time-step. Meaning, we do not have access to the "complete" hypotheses (over the entire horizon) in advance, but must "build" them gradually, by actually performing the belief updates.

Hence, we also cannot infer the involvement levels of the variables in these hypotheses ahead of time, but only infer the involvement levels in the upcoming myopic updates. Thus, we shall use PIVOT myopically and incrementally, when examining the multiple hypotheses of each candidate, by calculating and applying a PIVOT order *on the intermediate beliefs* at each time-step. In other words, we shall apply PIVOT before each "observation branching" of the hypothesis tree, as demonstrated in Fig. 5.5b.

We may recall that at each time-step, only the "first observation branching" determines the topology of the observations, i.e., which variables are involved in them (see Section 3.3.2). The observed value, which is determined in the "second branching", does not affect this aspect. Also, regardless of this value, any action to be applied after the observation, would only involve the last pose, by definition. Thus, we can indeed determine the myopic variable involvement levels (and the PIVOT order) between time-steps.

5.2.2 Heuristic Classification

In the previous two scenarios, we relied on the "true" variable classification, which we can derive unmistakably from the (accessible) future updates. It is also possible to try and heuristically *estimate* the variable involvement levels, even if we do not have full access to the future updates in advance. For example, we can use the "ML hypotheses" just to estimate the variable involvement, even if we consider planning with multiple hypotheses per candidate. We can also try to perform a precursory visibility or reachability analysis, to identify variables that are (almost) certainly involved or uninvolved in the future updates. In some cases, we may even know in advance that a certain subset of the state variables can never be involved by definition; e.g., past poses are never involved in landmark-based SLAM – where only landmarks can be re-observed.

In this case, we shall infer an apply a PIVOT order on the initial belief, in preparation for the long-horizon future updates, based on this estimated classification. We note that even if this variable classification does not match the "true" classification, inferred from the "real" updates (when they become available), this does cannot lead to any sacrifice in accuracy; at worst, such misclassification can lead to an increase in the computational cost of the updates.

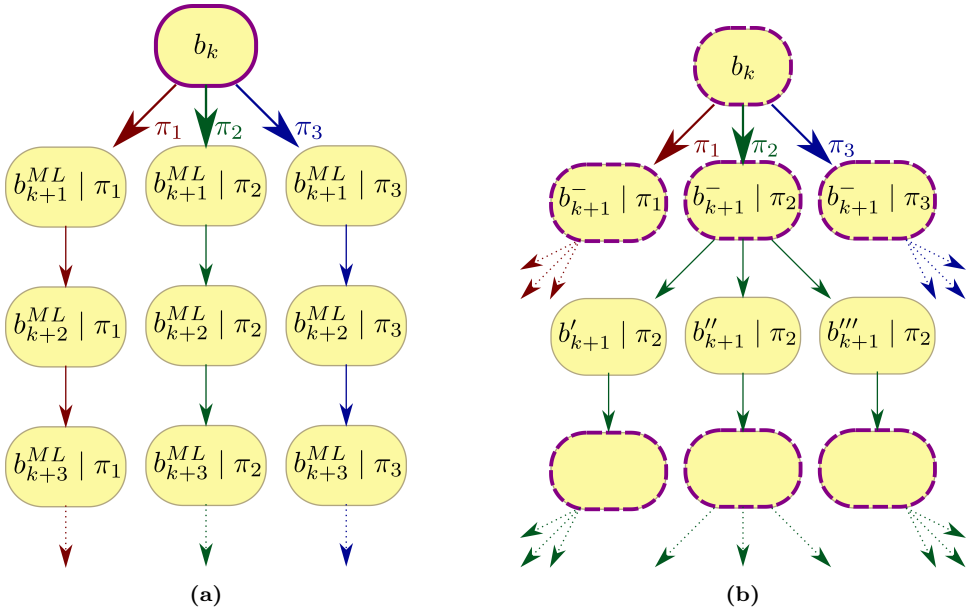


Figure 5.5: Planning trees, built by connecting the hypothesis trees (as presented in Fig. 3.6) of three candidate policies at the root (the initial belief). `PIVOT` is applied on belief nodes marked in purple.
(a) A planning tree when relying on ML observations. The entire tree is known ahead of time; hence, we shall apply `PIVOT` once on the initial belief, given the updates over the entire planning horizon.
(b) A planning tree with general (not ML) observations. The hypothesis trees are not fully known ahead of time, and we must calculate the belief updates in order to reveal the next tree levels; hence, we shall apply `PIVOT` on the beliefs at each time-step, given only the myopic updates (i.e., before each observation branching). It is also possible to apply `PIVOT` on the initial belief considering heuristic knowledge on the expected variable involvement levels.

5.3 Influence of PIVOT on State Inference and Re-Planning

Consider that in planning session at time-step k, we applied PIVOT on the *initial* belief (i.e., b_k). After planning and policy selection, we have the option to continue to the following inference and planning sessions with the belief in the original order (assuming it was cached), or continue with the belief in the PIVOT order. Let us examine the benefits (and possible shortcomings) of continuing with the modified order.

First, consider another re-planning session is due in the future. Since PIVOT can be applied incrementally, if the variable order is kept after the current planning session, then in the next planning session, the PIVOT order can simply be incrementally updated, according to the change in variable classification (and addition of new variables). In this case, the "fixed" variable class $\text{class}^0_{\mathcal{X}<j}$ would contain the uninvolved variables which kept their order and classification since the previous planning session; and $\text{class}^0_{\mathcal{X}>j}$ would contain the uninvolved variables which were shuffled forward in inference, and new uninvolved variables, whether they were added since the previous planning session, or simply changed their classification. Assuming the classification of variables usually does not change much between consecutive planning sessions, then keeping the PIVOT order shall make $\text{class}^0_{\mathcal{X}<j}$ bigger, and the cost of applying PIVOT again can be reduced.

Second, we can examine the influence on future state inference updates. As explained, incorporating updates to variables from the past (i.e., loop closures) is expensive, when these affect numerous variables. When applying PIVOT during a planning session, we push forward predicted loop closing variables, i.e., variables from the past, which we believe to be re-observed when executing the policy. We explained how doing so can reduce the computational cost of performing the predicted updates during planning. Now, if we are to actually perform the selected policy, it is likely that we indeed face the same (or similar) loop closures we have previously predicted, and have to update the belief accordingly. Hence, in this case, keeping the predicted loop closing variables forward (i.e., maintaining the PIVOT order after planning) can reduce the computational cost of the following state inference updates, just like it did during the (predicted) planning updates. As mentioned, when considering "pure" state inference, and ignoring the planning problem, there is no point in actively pushing forward loop closing variables, as the cost of such reordering is equivalent to actually performing the loop closure. However, since we conduct the reordering procedure anyway, as part of the planning, we gain this indirect benefit to the state inference process. This conclusion is derived from a wider view of the system, which considers both the state inference and planning processes.

Nonetheless, PIVOT is designed to benefit the overall update cost considering the identification of involved variables from the entire set of candidates; it is thus possible that the cost of specific updates would increase (in order to benefit the other updates). So, to decide if we should keep the PIVOT order, it is important to verify that the selected candidate – the one which we shall now apply – is not one of those specific updates. Thus, for the selected candidate, we shall check if the number of NZs (i.e., the density) in the affected block of the receptive update(s) under PIVOT is indeed smaller than under the original order. If this is the case, we shall continue with PIVOT. Note that in this verification, we implicitly assume that the prediction will coincide with the reality when performing the action. It is indeed possible that when performing the action, the actual realized hypothesis, and hence the updated variables, can be different than the predicted ones. However, there is no way to prepare for such a case, and our "best bet" is to assume the prediction represents the future reality (otherwise we should not have used it for planning in the first place).

Finally, we should also consider that PIVOT is intended to increase planning efficiency, in the ways indicated, and is not inherently fill-reducing. Typically, maintaining the constrained variable order after planning, would result in a sub-optimal fill-in in the belief during state inference, and increase the memory footprint. Also, although the cost of inference updates should actually improve, as we explained, the added fill-in in the square root matrix might increase the cost of back-substitution, which is solved when updating the state MAP estimate. The frequency of back-substitution in the system is a matter of design. For example, one may compute it after every state inference session, or only before planning. We also do not always need to calculate the entire back-substitution solution; updating the estimate of the last pose, which is usually at the end of the state, can be achieved almost immediately. Thus, overall, even if we expect some sacrifice in back-substitution cost, we expect the gains in performance to prevail.

5.4 PIVOT vs. Belief Sparsification

For good measure, let us compare PIVOT to belief sparsification, as formulated in Chapter 4. Clearly, both approaches are related, as they suggest to perform a certain modification to a high-dimensional belief, which serves as the initial belief of a decision problem. Since belief sparsification intrinsically requires a variable reordering step, the two methods (at times) have a similar influence of the belief update process.

Specifically, if we consider (1) sparsification of only uninvolved variables; (2) the sparsification algorithm (Algorithm 3) to operate on the square root matrix; and (3) no requirement to maintain the original variable order after the sparsification (by instead, reordering the collective Jacobians); then, the initial reordering is sufficient to make sure that the rows of the "sparsified" variables would not be updated when (incrementally) incorporating new constraints. Meaning, there is no need to actually zero entries in these rows, and this sparsification is computationally equivalent to application of $\texttt{PIVOT}_1$. Yet, the approaches are still different, as, from these basic variations, both approaches are "scalable" in different manners: sparsification – by selecting more variable for spasification; and PIVOT – by dividing the variables into more classes, and considering fill-aware optimizations.

Further, the two approaches are *logically* different, as sparsification conveys *approximation* of the belief, while reordering conveys merely a change of *representation*. By such, application of PIVOT (in all its variations) inherently does not influence the accuracy in any stage of the decision making process, and, specifically, does not compromise the quality of solution nor induces any loss. On the other hand, depending on the sparsified variables, calculating the objective values of the candidates using a sparsified belief may affect their accuracy, and by such compromise action consistency, and lead to simplification loss. As we saw, in that case, we should perform pre- or post-solution analysis to verify the quality of the yielded solution. Nonetheless, this sacrifice in accuracy can lead to additional increase in performance; performing "full" sparsification (potentially) leads to the highest solution efficiency, since it means planning with a "diagonal" (square root) information matrix.

Overall, although the basic variant of each approach ("uninvloved" sparsification and $\texttt{PIVOT}_1$) can have a similar effect on the updates, when extended, each approach enjoys it own unique strengths. To summarize, let $\succ$ mark the relation "better than"; then, in terms of efficiency:

$$\text{"full" sparsification} \succ \texttt{PIVOT}_c^* \succ \texttt{PIVOT}_1 \equiv \text{"uninvolved" sparsification} \succ \text{original solution};$$

and in terms of quality of solution:

$$\text{original solution} \equiv \texttt{PIVOT}_c/\texttt{PIVOT}_c^* \equiv \text{"uninvolved" sparsification} \succ \text{"full" sparsification};$$

Finally, we shall mention a few more advantages that are exclusive to planning with PIVOT, over belief sparsification: since the first is just a change of representation, we do not have to discard the modified belief after planning. This allows us to (i) improve the inference process, and (ii) efficiently update the order during re-planning; also, with PIVOT, unlike sparsification, (iii) we are not limited to information theoretic objectives.

5.5 Experimental Evaluation

To demonstrate the advantages of PIVOT, we applied it in the solution of **the same active-SLAM simulation described in Section 4.4**, which was previously solved using belief sparsification. Note that while the previous experiment focused solely on the solution of the planning sessions, here, we also examine the influence of PIVOT on the state inference sessions.

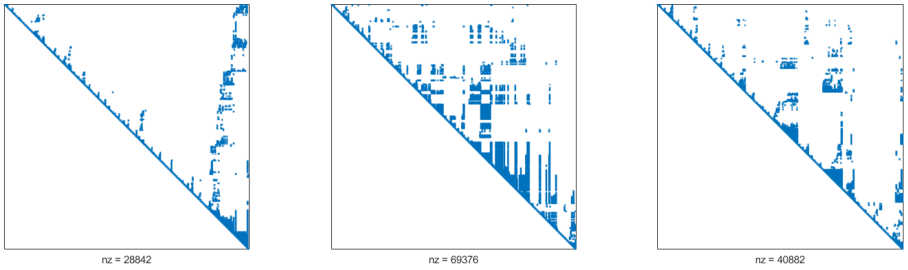
5.5.1 Utilizing PIVOT

In the considered scenario, the state size grows quickly as the navigation progresses, making planning more computationally challenging. Hence, we wish to examine how utilizing PIVOT can help reducing the planning cost. To do so, throughout the simulation, we maintained in parallel multiple versions of the (factorized) belief, using different ordering tactics, and performed the appropriate belief updates on all versions of the belief, during each planning and state inference session. We shall recall once again that changing the order of variables only conveys a change of representation, and does not affect the accuracy of the solution.

This time, the baseline ordering tactic we considered matches the one suggested by the state-of-the-art SAM algorithm iSAM2 (Kaess et al., 2012). According to this tactic, at each incremental update of the factorization, we shall calculate and apply a constrained fill-reducing order (CCOLAMD) on the affected variables; the constraint forces the involved variables forward, to the end of the state. While the reasons for applying a fill-reducing order as part of this tactic are obvious (reducing fill-in and the update cost), the added constraint leads to a sacrifice in fill-in, and does not contribute to the current inference session. However, the constraint is added with the intention to contribute to the following inference session: if the variables are to be updated again, then the affected block in the next session would be smaller. In their experiments, the authors demonstrated that this claim is well worthwhile for incremental state inference; however, it might not be the optimal one for planning.

Thus, we also considered utilizing (several variations of) PIVOT during planning. Since we rely on ML observations, this simply means performing an additional standalone reordering of the (initial) belief before calculating the objective values in each planning session. This comes in contrast to the baseline tactic, where (incremental) reordering is only performed during updates, and never as a standalone step. For comparison, we considered $\texttt{PIVOT}_1$, $\texttt{PIVOT}_5$, and $\texttt{PIVOT}_{\max}$, which are the basic PIVOT orders, with varying numbers of classes, and their fill-aware counterparts $\texttt{PIVOT}^*_1$, $\texttt{PIVOT}^*_5$, and $\texttt{PIVOT}^*_{\max}$. Although it is not obligatory, for this comparison, we chose to always continue to the following state inference sessions (after the planning is completed) with the modified orders; nonetheless, all updates during the state inference sessions were always conducted according to the baseline tactic. Overall, we maintained seven versions of the belief, according to seven ordering tactics: the baseline – in which we performed reordering only during updates, and six other tactics, in which we *additionally* applied a PIVOT variation in standalone reordering steps during planning sessions.

For each version, we measured and compared the total state inference and planning times. For each inference session, we measured the time of performing the belief factorization update and the back-substitution (to refine the MAP estimate); for each planning session, we measured



(a) Baseline $\boldsymbol{R}$. **(b)** $\boldsymbol{R}$ after applying $\texttt{PIVOT}_{\max}$. **(c)** $\boldsymbol{R}$ after applying $\texttt{PIVOT}^{*}_{\max}$.

Figure 5.6: A comparison of the sparsity pattern of three variants of the agent's square root matrix $\boldsymbol{R}$, at the final planning session.

the belief update time according to the predicted hypothesis of each candidate, along with the time of applying the `PIVOT` order, when it was used. As mentioned, when planning assuming ML observations, there is no need to perform back-substitution to infer the posterior state estimate.

Similarly to the previous experiment, from our GTSAM implementation, at each state inference session, we extracted the Jacobian rows, representing the newly added (linearized) factors, and performed (and timed) the incremental update of the belief's square root matrix, as formulated in (3.26), using the standard MATLAB implementation of `qr`. At each planning session, we extracted the collective Jacobian matrices representing the predicted hypotheses of each of candidates, and performed (and timed) a similar square root matrix update for each one. Of course, the columns of the extracted Jacobians were reordered appropriately before each update, to match the variable order of that version of the belief. The modification of the square root matrix in the variable reordering steps was performed according to Algorithm 6 (the "re-factorization" variant). The sparsity pattern of the square root matrices under different orders is presented in Fig. 5.6 (and analyzed in the next section); the candidates' collective Jacobian matrices can be re-examined in Section 4.4.

One might note that by extracting the new Jacobian rows from the GTSAM-based (non-linear and Gaussian) belief at each time step, we are not concerned by the possibility of re-linearization of the system in our analysis. In events of re-linearization, the belief's Jacobian matrix $\boldsymbol{J}$ (or parts of it) is recalculated, around an updated estimate, and must then be re-factorized (i.e., $\boldsymbol{R}$ must be recalculated). This means that re-linearization of the system (which, in our case, is handled internally by GTSAM) does not affect the belief update times we cared to measure in our experiment, as such an event would always happen *after* the update is completed. In fact, we could even use a re-linearization event to our advantage, by using it as an opportunity to reorder the system, and apply the new `PIVOT` order. Then, the cost of reordering would be "absorbed", and applying `PIVOT` would be "free". Since we do not wish to make assumptions on how often, and to which extent, one should perform re-linearization, we counted all reordering steps as if they were explicitly performed.

5.5.2 Results and Discussion

Next, we present and analyze the results of using each ordering tactic to solve the described scenario. Fig. 5.7 presents the accumulated planning time (belief updates and reordering), using each ordering tactic, throughout eight planning sessions. Fig. 5.8a similarly presents the accumulated state inference time (belief update and back-substitution), using each ordering tactic, throughout $(\sim)1,000$ inference sessions. Fig. 5.8b presents the number of non-zero entries in

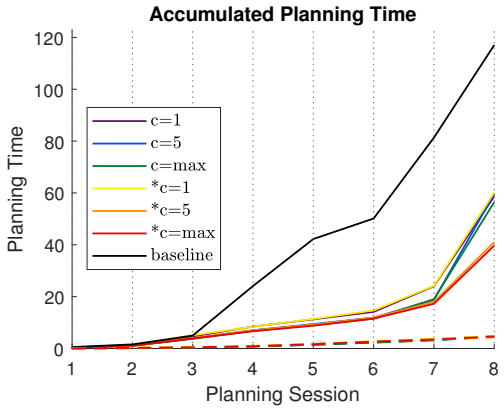


Figure 5.7: The accumulated planning time throughout the eight planning sessions of our experiment, using different variable ordering tactics: baseline ("iSAM2"), three variants of $\texttt{PIVOT}_c$, and three variants of $\texttt{PIVOT}_c^*$; lower is better. The dashed line represents the accumulated reordering time out of the measured time.

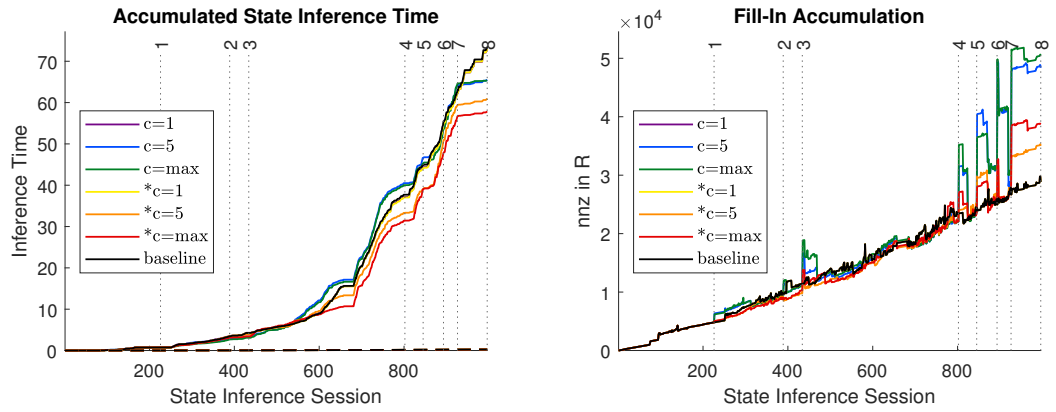
the square root matrix, which represents the fill-in of the factorization, in each state-inference session, using each ordering tactic. All the numerical results presented in these figures are also summarized in Table 5.1.

Several conclusions can be made: first, it is clear to see that using any variation of PIVOT managed to substantially reduce the planning time in comparison to baseline tactic ($50-66\%$ reduction), as intended. We can also see that the reordering time (i.e., time of applying the PIVOT order) gradually becomes minor, in comparison to the update time in each planning session, as the state grows. Further, all variations of PIVOT managed to reduce the state inference time – verifying our claims in Section 5.3 – and, by such, the total solution time of the scenario ($30-50\%$ reduction). To explain the contribution of PIVOT to the inference process, in comparison to the constrained baseline tactic, we may recall that with that tactic, we pushed forwards the loop closing variables only *after* they were met; when applying PIVOT, we pushed forwards (predicted) loop closing variables *before* meeting them, in the preceding planning sessions. This resulted in more efficient loop closing during state inference, compared to the baseline.

Of course, the relative improvement in planning time tightly depends on the number of candidates, the topology of their updates, and the size of the state; the improvement in total solution time is also a parameter of the planning to inference session ratio. Nonetheless, these results surely encourages us to utilize (a variation of) PIVOT in the solution of high-dimensional planning problems.

More specifically, we can see that each variation of PIVOT lead to different results. Generally, whether examining the per-session or accumulated results, the fill-aware variations of PIVOT resulted in lower planning time, inference time, and fill-in, in comparison to their "standard" counterparts; even the belief reordering time is lower in the fill-aware variations, as expected. Also, while, generally speaking, using PIVOT/PIVOT* led to increase in fill-in (number of NZs in $\boldsymbol{R}$), in comparison to the baseline, the fill-aware variants resulted in much lower fill-in, in comparison to their standard counterparts, as expected. These results encourages us to always utilize the PIVOT* over PIVOT.

We may also notice that the added fill-in led to a slight increase in the back-substitution time during the inference sessions; yet, the cost of this step is insignificant in relation to the update cost. As mentioned, Fig. 5.6 shows the difference in fill-in in three variations of the square root



(a) **(b)**

Figure 5.8: Comparing the solution of the state inference sessions in our experiment, using different variable ordering tactics: baseline ("iSAM2"), three variants of $\texttt{PIVOT}_c$, and three variants of $\texttt{PIVOT}_c^*$. (a) Accumulated state inference time throughout the (∼)1,000 inference sessions; lower is better. The dashed line represents the accumulated back-substitution time out of the measured time.
(b) Accumulated fill-in (number of non-zeros in $\boldsymbol{R}$).

matrix, at the final planning session: the baseline, after the applying $\texttt{PIVOT}_{\max}$, and $\texttt{PIVOT}_{\max}^*$.

In most cases, increasing the number of classes (as determined by the parameter c), lead to lower planning and inference times, as this caused the involved variables to be pushed "further forwards". Yet, with both PIVOT and PIVOT*, this also resulted in increased fill-in. When using only a single PIVOT class (i.e., $c = 1$), we can see that the fill-in was hardly impacted. In fact, by applying $\texttt{PIVOT}_1^*$ in the final planning session, we even managed to reduce the fill-in below the baseline level. Interestingly, we can see that in the last planning session, the added fill-in induced by $\texttt{PIVOT}_5$ and $\texttt{PIVOT}_{\max}$ was so significant, that it actually made the planning cost higher, in comparison to the baseline; this might occur when the number of NZs (NNZ) in the affected block (of a candidate update) grows more significantly than the decrease in block size. Fortunately, this is not an issue when considering the fill-aware PIVOT* tactics. In fact, it is evident that $\texttt{PIVOT}_{\max}^*$ achieved the best performance in both planning and inference.

We can see that between planning sessions 3 and 4, which were seprated by numerous state inference sessions, the fill-in induced by the PIVOT variants decreased back to the baseline level. It appears that the incremental reordering that takes place in those inference sessions has gradually "canceled out" the PIVOT constraints, and realigned the order back to the baseline. In contrast, in the latter planning sessions, which occurred more frequently in between the inference sessions, we can see that the fill-in accumulated. Overall, we may conclude that frequent re-planning, and/or a higher number of classes, can amplify both PIVOT's disadvantages (relative added fill-in), and benefits (relative reduction in computation time).

As mentioned, here we maintained the PIVOT order(s) after every planning session. Although we saw this can indeed help reducing the state inference cost, if we wanted, we could also revert back to the original variable order (before the standalone reordering) after each planning session. That way, the state inference process (and the fill-in in $\boldsymbol{R}$) is not influenced by PIVOT, which only affects the planning process. As explained in Section 5.3, we can also analyze the fill-in in the affected block of the selected action, to verify that the state inference is indeed expected to improve by keeping the PIVOT order, before making such decision.

Table 5.1: Summary of the accumulated planning time (as presented in Fig. 5.7), inference times (as presented in Fig. 5.8), and fill-in at the final planning session (as presented in Fig. 5.6). Best values in **bold**.

Tactic		Baseline	PIVOT$_c$			PIVOT$_c^*$		
# of classes		N/A	1	5	max	1	5	max
Planning Session #1	Reordering time	/	00.05	00.06	00.06	00.07	00.08	00.07
	Update time	0.56	00.39	00.30	00.30	00.26	00.29	**00.28**
Planning Session #2	Reordering time	/	00.11	00.06	00.06	00.26	00.20	00.16
	Update time	01.02	00.64	00.64	00.56	00.57	00.52	**00.52**
Planning Session #3	Reordering time	/	00.05	00.21	00.22	00.23	00.18	00.18
	Update time	03.40	03.36	**02.52**	02.53	03.72	02.78	02.64
Planning Session #4	Reordering time	/	00.63	00.45	00.45	00.48	00.42	00.45
	Update time	19.12	03.20	02.77	02.56	02.78	02.54	**02.33**
Planning Session #5	Reordering time	/	00.90	00.66	00.59	00.76	00.79	00.75
	Update time	18.12	01.88	01.71	01.71	02.22	01.37	**01.35**
Planning Session #6	Reordering time	/	00.84	00.85	00.88	01.02	01.02	01.03
	Update time	07.90	02.00	**01.61**	01.54	02.27	01.69	01.70
Planning Session #7	Reordering time	/	01.01	00.92	00.86	00.89	00.79	00.78
	Update time	31.20	08.89	05.92	06.67	08.58	05.26	**04.95**
Planning Session #8	Reordering time	/	01.15	01.40	01.42	01.09	00.75	01.20
	Update time	35.70	33.49	39.52	35.98	34.91	22.30	**21.23**
Total planning time	Sec.	117.05	58.69	59.68	56.45	60.21	41.07	**39.70**
	Relative	100%	50.14%	50.98%	48.22%	51.43%	35.08%	**33.91%**
Total inference time	Update time	73.19	72.60	65.27	65.08	72.69	60.71	**57.62**
	Back-sub. time	**00.24**	00.25	00.28	00.31	00.24	00.25	00.26
Total solution time	Sec.	190.49	131.55	125.24	121.82	133.16	102.05	**97.57**
	Relative	100%	69.00%	65.74%	63.95%	69.90%	53.57%	**51.22%**
Fill-in (NNZ in $\boldsymbol{R}$)	Before reordering	**28842**	**28842**	48372	50472	**28842**	35029	38749
	After reordering	/	31571	62091	69376	**27662**	32737	40882

Chapter 6

Modification of the Square Root Matrix on Variable Reordering

As covered, utilization of belief sparsification (Chapter 4) and application of `PIVOT` (Chapter 5) require to perform reordering of the state variables. As we learned, in the practical linear(ized) and Gaussian case, the order of variables must match the order of columns in the coefficient matrix of the system, which we factorize to derive the "square root matrix". Hence, reordering the variables implies expensive re-factorization of the system (under the new variable order), in order to derive the "new" square root matrix. Instead, in this chapter, we bring a new algorithm, which allows to efficiently modify an existing square root matrix to appropriately convey reordering of the variables.

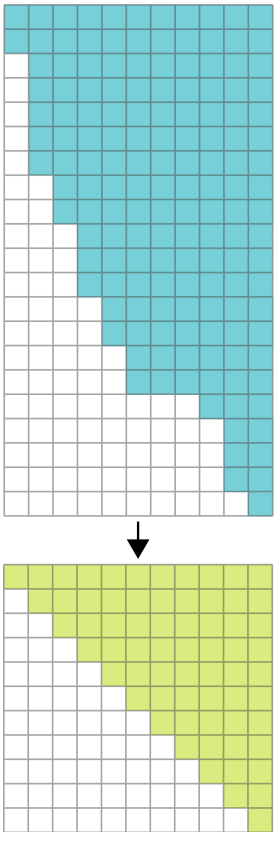


Figure 6.1: $\boldsymbol{J}$ (top) and $\boldsymbol{R}$, given from $\mathtt{qr}\{\boldsymbol{J}\}$.

6.1 Problem Definition and Contribution

We formulate the problem of interest as follows: consider the coefficient matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of a system of linear equations (in our context, this is the "Jacobian matrix"), and its upper triangular factor $\boldsymbol{R}$ (the "square root matrix"), given by the QR factorization, which satisfies $\boldsymbol{J} = \boldsymbol{Q}\boldsymbol{R}$, for some orthogonal matrix $\boldsymbol{Q}$. Also consider a column permutation p. Formally, this is a bijective function $\{1, \dots, n\} \to \{1, \dots, n\}$, which defines a mapping of each column (state variable) from index i to index $p(i)$. Such a permutation can be described with an orthogonal permutation matrix $\boldsymbol{P}^{n \times n}$, where $\boldsymbol{e_i} \in \mathbb{R}^n$ is the i-th unit vector:

$$\boldsymbol{P} \doteq \begin{bmatrix} | & & | \\ \boldsymbol{e_{p(1)}} & \cdots & \boldsymbol{e_{p(n)}} \\ | & & | \end{bmatrix}, \tag{6.1}$$

We wish to find the upper triangular factor $\boldsymbol{R_p}$ of the permuted matrix $\boldsymbol{J_p} \doteq \boldsymbol{J}\boldsymbol{P}$, which satisfies $\boldsymbol{J_p} = \boldsymbol{Q_p}\boldsymbol{R_p}$, for some orthogonal matrix $\boldsymbol{Q_p}$. As explained, we may equivalently consider $\boldsymbol{R}$ and $\boldsymbol{R_p}$ to be the Cholesky factors of $\boldsymbol{\Lambda} \doteq \boldsymbol{J}^T\boldsymbol{J}$ and $\boldsymbol{\Lambda_p} \doteq \boldsymbol{P}^T\boldsymbol{\Lambda}\boldsymbol{P}$, respectively.

For visualization purposes, we consider an exemplary system, which will be used as a running example throughout the chapter. In the top sub-figure of Fig. 6.1, we can see the original coefficient matrix $\boldsymbol{J}$ of the system, and underneath it, its factor given from $\mathtt{qr}\{\boldsymbol{J}\}$. We wish to apply the variable permutation p, such that

$$[\,1,\ldots,11\,]\cdot\boldsymbol{P}\doteq[\,1,2,4,3,8,9,5,7,6,10,11\,]. \tag{6.2}$$

Each of the sub-figures represents the sparsity pattern of the corresponding matrix; colored cells represent entries which are (possibly) Non-Zero (NZ); different colors correspond to the type of entry, as to be described. For convenience, the rows of $\boldsymbol{J}$ are sorted according to the column-index of their leading NZ entry (i.e, first NZ in the row); nonetheless, $\boldsymbol{R}$ is invariant under row reordering in $\boldsymbol{J}$. We do not make any assumptions on the density (or sparsity) of these matrices.

To clarify, we do not wish to choose a new variable order; we consider it to be given, and address how to perform the appropriate modification to $\boldsymbol{R}$ practically and efficiently. To achieve this goal, we derive three key contributions: first, we identify that variable reordering only requires a local modification of $\boldsymbol{R}$, in a contained block of rows; second, we explain how to identify independent row blocks in $\boldsymbol{R}$, which can be modified independently, and in parallel; third, we explain how to efficiently derive the orthogonal transformation, which defines the required modification for each row block. We then utilize these three ideas in two algorithmic variants: one, by manipulating the factor directly, without accessing $\boldsymbol{J}$; and second, with partial re-factorization of $\boldsymbol{J}$. Each of the variants holds specific advantages, as we shall compare.

6.2 Direct Modification

6.2.1 The Local Effect of Variable Reordering

First, we shall try to derive $\boldsymbol{R_p}$ by appropriately modifying the original square root matrix $\boldsymbol{R}$, and without accessing $\boldsymbol{J}$. Let us look at $\boldsymbol{RP}$, i.e., $\boldsymbol{R}$ with columns permuted according to p. As visualized in the left sub-figure of Fig. 6.2, applying the column permutation breaks the triangular shape of the original root matrix $\boldsymbol{R}$. We may re-apply $\mathtt{qr}$ to "correct" it:

$$\{\boldsymbol{Q'},\boldsymbol{R'}\}\doteq\mathtt{qr}\{\boldsymbol{RP}\} \tag{6.3}$$

The permutation does not affect the rank of the matrix, and since $\boldsymbol{R}$ is of full rank (the rows/columns are linearly independent), so is $\boldsymbol{RP}$. Therefore, we know a *unique* $\mathtt{qr}$ factorization of it exists. A similar re-application of $\mathtt{qr}$ to "correct" the factor is performed also by the aforementioned iSAM algorithm; though, there, the factor is modified due to the addition of constraints (new rows), and not permutation of the state variables.

This factorization can be calculated in several manners. We may choose to apply a series of Givens rotations $\{\boldsymbol{Q_k}\}_k$ to zero all elements below the diagonal, one entry at a time, column-by-column. Each of these rotations is applied only on a pair of rows in the factorized matrix; the angle of rotation is chosen such that the leading NZ entry in the lower of the two rows is zeroed (for more details, see Golub and Loan (1996)). Since the factorization exists, we know that after a finite number of such rotations, we will end up with an upper triangular matrix $\boldsymbol{R'}$ with a NZ diagonal, such that

$$\boldsymbol{R'}=\boldsymbol{Q'}^T\cdot(\boldsymbol{RP}),\ \text{ where }\ \boldsymbol{Q'}^T\doteq\prod_k\boldsymbol{Q_k}. \tag{6.4}$$

Surely, since the product of orthogonal matrices remains orthogonal, $\boldsymbol{R'}$ is the desired square root matrix of the permuted system:

$$\boldsymbol{J_p}=(\boldsymbol{QR})\cdot\boldsymbol{P}=\boldsymbol{Q}\cdot(\boldsymbol{RP})=\boldsymbol{Q}\cdot(\boldsymbol{Q'R'})=(\boldsymbol{QQ'})\cdot\boldsymbol{R'}. \tag{6.5}$$

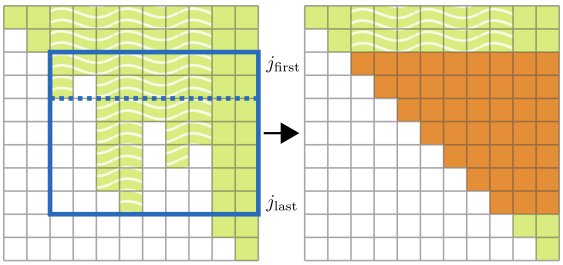


Figure 6.2: The local effect of variable reordering. Blue borders mark the re-factorized sub-matrix; green-colored entries represent entries of $\boldsymbol{R}$; orange represents entries calculated from re-factorization of $\boldsymbol{RP}$; white overlay indicates column permutation. Modification of $\boldsymbol{R}$ only requires factorization of the block of rows $j_{\text{first}} : j_{\text{last}}$ of $\boldsymbol{RP}$. Using Algorithm 5, we can divide this row block (dashed blue line) into the two sub-blocks, which can be factorized independently.

Although it provides the solution, we do not suggest to use this naive approach to modify $\boldsymbol{R}$, as it may be far from optimal. Yet, due to the uniqueness of $\boldsymbol{R'}$, we know that this $\boldsymbol{R'} \equiv \boldsymbol{R_p}$, and we can use this solution to derive general properties $\boldsymbol{R_p}$, which we will exploit to derive it more efficiently.

Let us mark with j_{first} and j_{last} the the minimal and maximal non-fixed points of p, respectively; i.e., the minimal and maximal indices $j \in 1 : n$, for which $p(j) \neq j$. NZ entries below the diagonal can only appear in $\boldsymbol{RP}$ in rows indexed between j_{first} and j_{last}. Hence, application of the aforementioned Givens rotations shall only modify these rows of the factorized matrix; $\boldsymbol{R'}$ ($\equiv \boldsymbol{R_p}$) remains equal to $\boldsymbol{RP}$ in all other rows. The underlying conclusion, which holds regardless of the manner by which $\boldsymbol{R_p}$ is calculated, is conveyed in Proposition 1.

Proposition 1 (The Local Effect of Variable Reordering)**.**
The block of rows $j_{first} : j_{last}$ of $\boldsymbol{R_p}$ relies only on the respective block of rows of $\boldsymbol{R}$; the matrices may differ in value only in these rows. The top $(j_{first} - 1)$ rows, and bottom $(n - j_{last})$ rows of $\boldsymbol{R_p}$, are identical to the respective rows of $\boldsymbol{R}$ (up to column permutation).

Thus, $\boldsymbol{R_p}$ may be derived using any `qr` implementation as:

$$\boldsymbol{R_p} = \begin{bmatrix} — & \boldsymbol{R'}_{\text{top}} & — \\ \boldsymbol{0} & \boldsymbol{R'}_{\text{middle}} & \\ — & \boldsymbol{R'}_{\text{bottom}} & — \end{bmatrix}, \text{ where} \tag{6.6}$$

$$\boldsymbol{R'}_{\text{top}} \doteq \boldsymbol{RP}(1 : (j_{\text{first}} - 1), 1 : n), \tag{6.7}$$

$$\{\boldsymbol{Q'}_{\text{middle}}, \boldsymbol{R'}_{\text{middle}}\} \doteq \texttt{qr}\{\boldsymbol{RP}(j_{\text{first}} : j_{\text{last}}, j_{\text{first}} : n)\}, \tag{6.8}$$

$$\boldsymbol{R'}_{\text{bottom}} \doteq \boldsymbol{RP}((j_{\text{last}} + 1) : end, 1 : n). \tag{6.9}$$

Accordingly, $\boldsymbol{Q'} \doteq \texttt{blkdiag}\{\boldsymbol{I}_{\boldsymbol{j}_{\text{first}}-\boldsymbol{1}}, \boldsymbol{Q'}_{\text{middle}}, \boldsymbol{I}_{\boldsymbol{m}-\boldsymbol{j}_{\text{last}}}\}$ marks the "correcting" orthogonal matrix, such that $\boldsymbol{Q_p} = \boldsymbol{Q} \cdot \boldsymbol{Q'}$ completes the solution. This "localized" direct modification method is visualized in Fig. 6.2.

We may also reach a similar conclusion from probabilistic analysis. The belief $b(\boldsymbol{X})$ (which we defined in Section 3.1.1) can be factorized to a product of conditional probabilities:

$$b(\boldsymbol{X}) \propto \prod_{k=1}^{n-1} \mathbb{P}(\boldsymbol{X}(k) | d(\boldsymbol{X}(k))) \cdot \mathbb{P}(\boldsymbol{X}(n)), \tag{6.10}$$

where $d(x)$ denotes the set the variables x is conditioned on, for a given variable elimination order. Now, say we wish to modify this factorization to match a permuted elimination order, which is defined by the permutation p. By definition, $d(\boldsymbol{X}(j)) = d(\boldsymbol{X}(p(j)))$ for $j < j_{\text{first}}$ and $j > j_{\text{last}}$. Thus,

$$b(\boldsymbol{X}) \propto \prod_{j=j_{\text{first}}}^{j_{\text{last}}} \mathbb{P}(\boldsymbol{X}(p(i))|d(\boldsymbol{X}(p(i)))) \cdot \prod_{\substack{1\leq j<j_{\text{first}} \\ j_{\text{last}}<j\leq(n-1)}} \mathbb{P}(\boldsymbol{X}(k)|d(\boldsymbol{X}(x))) \cdot \mathbb{P}(\boldsymbol{X}(n)) \tag{6.11}$$

Meaning, the permutation only affects the conditional probabilities of the variables $j_{\text{first}} : j_{\text{last}}$, or, in other words, permutation has only a local effect on the belief factorization – with no assumptions on the type of distribution! Nonetheless, we recall that, when $\boldsymbol{X}$ is normally distributed, this factorization of $b(\boldsymbol{X})$ corresponds to factorization of $\boldsymbol{J}$ to the matrix $\boldsymbol{R}$, where each conditional probability corresponds to the respective row of $\boldsymbol{R}$ (see Section 3.2.1). Hence, in accordance with Proposition 1, we may again conclude that applying the permutation only leads to modification of rows $j_{\text{first}} : j_{\text{last}}$ of this matrix.

6.2.2 Identifying Independent Row Blocks

It is known that every permutation can be decomposed into disjoint cycles. Each cycle defines a "sub-permutation" which only affects a distinct subset of the input elements (in our case, the indices $1 : n$), while all other elements are stationary (i.e., mapped to themselves). We can write our exemplary permutation (6.2) in "cycle notation":

$$p = (1)(2)(3\,4)(5\,8\,7)(6\,9)(10)(11). \tag{6.12}$$

Each set of parentheses represents a disjoint cycle; each number in the cycle is mapped to the one that follows it, and the last number is mapped back to the first one.

Thanks to Proposition 1, we know that applying each such "sub-permutation" only requires factorization of a specific block of rows of $\boldsymbol{RP}$, encapsulated between the minimal and maximal index in the cycle, independently of the other rows. However, we note that multiple "sub-permutations" cannot always be applied independently of each other, as these blocks of rows, corresponding to each of the cycles, are not always distinct. This happens when the cycles define overlapping ranges, like $(5\,8\,7)$ and $(6\,9)$. In this case, the two cycles cannot be considered independently, and we have to factorize the entire block of rows $5 : 9$. While accounting for such scenarios, Algorithm 5 specifies how to divide the rows of $\boldsymbol{RP}$ into distinct row blocks, which can be treated independently, given a certain permutation. The result of this algorithm is visualized in Fig. 6.2, where, given p, we identified two row blocks of $\boldsymbol{RP}$. After the blocks are identified, we can "correct" each of them by applying an independent `qr` factorization. These factorizations can (and should) be executed *in parallel*.

6.2.3 Efficient Row Modification

Assume we wish to factorize the non-triangular block of rows $j_1 : j_2$ of $\boldsymbol{RP}$, in order to derive the respective block of rows of $\boldsymbol{R_p}$. We note that, even without the leading zero columns, this is a "wide" matrix, i.e., the number of columns is greater than or equal the number of rows. Hence, let us examine a sub-matrix of this row block, containing only its first $l \doteq j_2 - j_1 + 1$ NZ columns: $square\_block \doteq \boldsymbol{RP}(j_1 : j_2, j_1 : j_2)$. This is a square block taken around the diagonal of $\boldsymbol{RP}$. This block is of full rank l, as it differs from the block $\boldsymbol{R}(j_1 : j_2, j_1 : j_2)$ only in the order

Algorithm 5: Identifying independent row blocks.

Inputs:
Permutation p

Output:
block_division – a set of vectors, each containing row (variable) indices of an independent row block in $\boldsymbol{R_p}$

```
1  block_division ← {}
2  j1 ← 1
3  while j1 ≤ n do
4      j2 ← p(j1)
5      if j2 > j1 then
           // not a stationary point, look for intersecting cycles
6          k ← j1 + 1
7          while k ≤ j2 do
8              j2 ← max{j2, p(k)}
9              k ← k + 1
10     block_division ← block_division ∪ {j1 : j2}
11     j1 ← j2 + 1
```

of columns, and the latter is upper triangular with NZs on its diagonal (and hence of full rank). Therefore, *square_block* has a unique factorization:

$$\{\boldsymbol{Q'}_{\text{block}}, square\_block'\} = \texttt{qr}\{square\_block\}, \tag{6.13}$$

such that *square_block*′ is also an upper triangular matrix with NZs on its diagonal. Thus, it is sufficient to examine *square_block*, in order to derive the matrix $\boldsymbol{Q'}_{\text{block}}$, which defines the desired transformation for the entire row block.

This approach essentially separates the calculation and application of this orthogonal transformation; this can improve the modification efficiency in comparison to a "naive" application of `qr` on the entire row block. Calculating the `qr` factorization requires finding and applying an appropriate series of transformations (e.g., Givens rotations or Householder reflections (Golub and Loan, 1996)), which together form the orthogonal matrix. Generally, we must apply each rotation, in order to find the next one in the series. This means that each entry in the factorized matrix is modified multiple times throughout the factorization, according to the number of transformations needed. Thus, by only factorizing the square block, and not the entire rows, we can avoid unneeded calculations. The entries beyond column j_2 (i.e., $\boldsymbol{RP}(j_1 : j_2, (j_2+1) : n)$) do not need to be continuously modified throughout the factorization process; they can be modified only once, by multiplying them in the matrix ${\boldsymbol{Q'}_{\text{block}}}^T$ after it is found.

Overall, we shall repeat this process for every row block of $\boldsymbol{RP}$ which requires factorization (provided by Algorithm 5); by doing so, we can identify a block diagonal structure, such that all the NZ entries of $\boldsymbol{RP}$ below the diagonal are contained (as visualized in Fig 6.3a). In order to calculate $\boldsymbol{R_p}$, we shall (independently) factorize each of these square blocks, and apply the transformations to the rest of the respective row blocks. As mentioned, $\boldsymbol{Q_p} = \boldsymbol{Q} \cdot \boldsymbol{Q'}$, and $\boldsymbol{Q'}$ can be found by concatenating each block's $\boldsymbol{Q'}_{\text{block}}$ matrix, according to the block diagonal structure. Algorithm 6 summarizes the approach.

Algorithm 6: Optimized factor modification.

Inputs:
- Upper triangular factor $\boldsymbol{R}$
- Permutation p with a matching column permutation matrix $\boldsymbol{P}$
- **For re-factorization:** $\boldsymbol{J}$ and *marginals* (see Section 3.2.1)

Output:
- Modified factor $\boldsymbol{R_p}$

$\boldsymbol{RP} \leftarrow \boldsymbol{R} \cdot \boldsymbol{P}$
$\boldsymbol{JP} \leftarrow \boldsymbol{J} \cdot \boldsymbol{P}$; `// *only for re-factorization*`
$\boldsymbol{R_p} \leftarrow zeros(n, n)$; `// init. matrix`
block_division $\leftarrow$ run Algorithm 5 on p
foreach *block_indices* **in** *block_division* **do** `// in parallel`
  $j_1 \leftarrow block\_indices(1)$
  $l \leftarrow length\{block\_indices\}$
  if $l == 1$ **then**
    $\boldsymbol{R_p}(j_1, j_1 : n) \leftarrow \boldsymbol{RP}(j_1, j_1 : n)$
  else
    $j_2 \leftarrow block\_indices(end)$
    if *DIRECT MODIFICATION* **then**
      $row\_block \leftarrow \boldsymbol{RP}(j_1 : j_2, j_1 : n)$
    else if *RE-FACTORIZATION* **then**
      $row\_block \leftarrow \begin{bmatrix} marginals_p\{j_1 - 1\} \\ \boldsymbol{JP}(\mathtt{I}_{\boldsymbol{JP}}\{j_1 : j_2\}, j_1 : n) \end{bmatrix}$
    `// calculate transformation`
    $square\_block \leftarrow row\_block(1 : end, 1 : l)$
    $\{Q'_{\text{block}}, square\_block'\} \leftarrow \mathtt{qr}\{square\_block\}$
    `// apply transformation`
    $rows' \leftarrow {Q'_{\text{block}}}^T \cdot row\_block(1 : end, (l + 1) : end)$
    `// assign in the modified matrix`
    $\boldsymbol{R_p}(j_1 : j_2, j_1 : j_2) \leftarrow square\_block'(1 : l, 1 : end)$
    $\boldsymbol{R_p}(j_1 : j_2, (j_2 + 1) : end) \leftarrow rows'(1 : l, 1 : end)$

6.2.4 Numerical Concerns

We recall that $\boldsymbol{R_p}$ is unique (under the aforementioned conditions), and, thus, all modification methods should theoretically yield the same matrix. However, the orthogonal transformations used in the factorization are operations with real numbers, which are inevitably rounded, in order to be computed in a machine with limited precision. Therefore, two mathematically equivalent sequences of operations might, in practice, reach different results, due to slight numerical errors.

Altogether, we can divide the modification methods into two categories: direct modification of $\boldsymbol{R}$, without re-accessing $\boldsymbol{J}$ (as considered thus far); and methods which, to varying capacity, involve re-factorization of $\boldsymbol{J}$, under the new variable order. Direct modification tends to be faster, as it takes advantage of the original factorization, in which we handled the "heavy load" of reducing the overdetermined system to a "square" one. When performing direct modification, we essentially apply additional transformations *on top* of those which were applied in the original factorization: $\boldsymbol{R_p} = \boldsymbol{Q'}^T \cdot \boldsymbol{Q}^T \cdot \boldsymbol{JP}$. Hence, as a rule of thumb, this approach corresponds to an overall longer (although theoretically equivalent) series of operations to reach $\boldsymbol{R_p}$, than if we were to re-factorize $\boldsymbol{JP}$ ($\boldsymbol{J} \rightarrow \boldsymbol{R} \rightarrow \boldsymbol{R_p}$ vs. $\boldsymbol{J} \rightarrow \boldsymbol{R_p}$). This may cause the matrix returned by

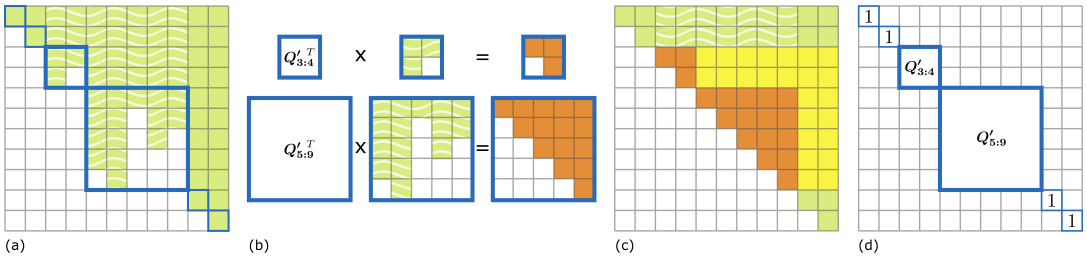


Figure 6.3: Applying Algorithm 6 for optimized direct modification of the factor $\boldsymbol{R}$ given p. Blue borders mark the re-factorized sub-matrices; green-colored entries represent entries of $\boldsymbol{R}$; orange represents entries calculated from re-factorization of $\boldsymbol{RP}$; yellow represents entries calculated by applying a pre-calculated transformation; white overlay indicates column permutation. (a) $\boldsymbol{RP}$ and its block diagonal structure. (b) Applying `qr` independently on each square block. (c) Deriving $\boldsymbol{R_p}$ by applying the calculated orthogonal transformations on the remaining elements in the respective rows. (d) The "correcting" orthogonal matrix $\boldsymbol{Q'}$ follows the same block structure.

direct modification to be more prone to numerical errors, and, most importantly, to "false" fill-in. Such errors occur when a series of operations that should accumulate to a zero entry, yields a slightly different value. This added fill-in can be problematic in sparse systems or sequential inference, as it can accumulate over time. As each of the approaches has its advantages (as we shall compare), we follow the previous discussion by addressing modification via re-factorization.

6.3 Modification via Re-factorization

Clearly, as demonstrated in Fig. 6.4a, we can derive $\boldsymbol{R_p}$ by simply re-factorizing the system under the new variable order:

$$\{\boldsymbol{Q_p}, \boldsymbol{R_p}\} = \texttt{qr}\{\boldsymbol{JP}\}. \tag{6.14}$$

In the top sub-figure, we can see the permuted coefficient matrix $\boldsymbol{J_p}$, and in the bottom one – its triangular factor. Yet, this "naive" solution is far from efficient; as we can see, this solution is oblivious to the original factorization, and all entries of $\boldsymbol{R_p}$ are calculated from scratch. As we shall formulate, it is possible to take advantage of the original factorization, and perform smart partial re-facatorization of $\boldsymbol{J_p}$.

6.3.1 Optimized Re-factorization

As covered in Chapter 3.2, state-of-the-art approaches for incremental smoothing and mapping perform incremental updates to $\boldsymbol{R}$, at the arrival of new constraints (i.e., addition of rows to $\boldsymbol{J}$). Although we do not consider addition of constraints, but only reordering of $\boldsymbol{J}$'s columns, we can still utilize this idea here, to efficiently calculate $\boldsymbol{R_p}$ from $\boldsymbol{JP}$, as visualized in Fig. 6.4b. Of course, we recall that such calculation requires caching of the marginals in the original factorization process. Nonetheless, this method is not optimized for pure variable permutation, as we consider here; we can improve it by utilizing our previous conclusions. From Proposition 1, we know that not only the top rows of $\boldsymbol{R}$ are preserved in the permutation, but also the bottom ones. So, similarly to (6.8), we can also "skip" the factorization of rows $\texttt{I}_{\boldsymbol{JP}}\{j_{\text{last}}+1:n\}$ of $\boldsymbol{JP}$, and factorize only rows $\texttt{I}_{\boldsymbol{JP}}\{j_{\text{first}}:j_{\text{last}}\}$, which, together with the permuted marginal factors,

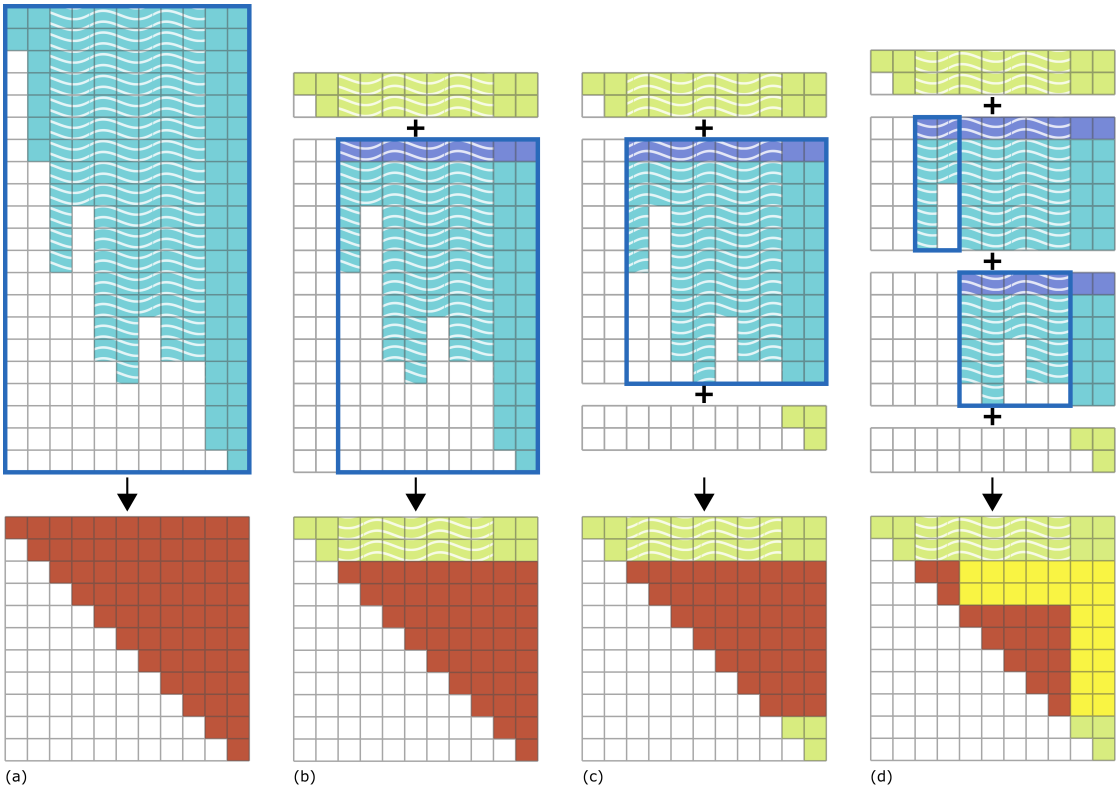


Figure 6.4: Modification via re-factorization. Blue borders mark the re-factorized sub-matrices; teal-colored entries represent entries of $\boldsymbol{J}$ (i.e., factors); green represents entries of $\boldsymbol{R}$; purple represents marginal factors; red represents entries calculated from re-factorization of $\boldsymbol{J}$; yellow represents entries calculated by applying a pre-calculated transformation; white overlay indicates column permutation. (a) Naive/complete re-factorization. (b) iSAM2-style incremental factorization, starting from the first permuted variable. (c) Utilizing the local effect of variable reordering. (d) Optimized re-factorization, as described in Algorithm 6; utilizing block independence and efficient block modification; the highlighted blocks can be factorized in parallel.

yield rows $j_{\text{first}} : j_{\text{last}}$ of $\boldsymbol{R_p}$, i.e.,

$$\mathtt{qr}\left\{\begin{bmatrix} marginals_p\{j_{\text{first}}-1\} \\ \boldsymbol{JP}(\mathtt{I}_{\boldsymbol{JP}}\{j_{\text{first}} : j_{\text{last}}\}, j_{\text{first}} : n) \end{bmatrix}\right\}. \tag{6.15}$$

This improvement is visualized in Fig. 6.4c. We note that here, unlike in direct modification, it is possible that the triangular block resulting from this re-factorization would contain additional rows, representing residual marginals factors; such rows can be trimmed, to be left only with the relevant rows of $\boldsymbol{R_p}$.

Furthermore, we can also utilize the concept of block independence, and divide rows $\mathtt{I}_{\boldsymbol{JP}}\{j_{\text{first}}, j_{\text{last}}\}$ of $\boldsymbol{JP}$ into smaller independent row blocks, which can be factorized in parallel – just like we considered in direct modification. Here, though, to every row block we shall add its relevant marginal factors (which were pre-cached). Finally, in each of these row blocks, we can separate the calculation of the orthogonal transformation from its application, as previously suggested. For a block of rows of $\boldsymbol{JP}$ (with the marginal factors), representing l rows of $\boldsymbol{R_p}$, it

is sufficient to examine its first l columns, in order to calculate the desired transformation. Only then we shall apply it to the remaining columns of that row block. We note, though, that in this case, this sub-block is not necessarily square, as the number of rows to re-factorize may be greater than the rank of the block. Application of these two concepts is visualized in Fig. 6.4d, and the overall optimized modification method is summarized in Algorithm 6. Also, we recall that formation of the orthogonal matrix $\boldsymbol{Q_p}$ is often not needed, as the transformations can be directly applied on the system's (cached) RHS vector, during the factorization. If still needed, we can derive $\boldsymbol{Q_p}$ from its blocks through appropriate row-index tracking. Alternatively, once $\boldsymbol{R_p}$ is found, $\boldsymbol{Q_p}$ can be easily found as well, by calculating $\boldsymbol{Q'} = \boldsymbol{RP} \cdot {\boldsymbol{R_p}}^{-1}$ (where $\boldsymbol{Q_p} = \boldsymbol{Q} \cdot \boldsymbol{Q'}$).

6.4 Experimental Results

We would now like to demonstrate the advantages of our optimized modification technique, provided in Algorithm 6, in a realistic scenario. Therefore, we again solved a sequential SLAM simulation (as brought in detail in Section 4.4), in which we estimated the trajectory of a ground robot navigating in an unknown environment. At the end of the trajectory, the state vector contained nearly 400 (three-dimensional) poses. We then extracted the coefficient (Jacobian) matrix $\boldsymbol{J}$, and the square root matrix $\boldsymbol{R} = \mathtt{qr}\{\boldsymbol{J}\}$, to be used in this demonstration; the sparsity pattern of these matrices can be viewed in Figs. 6.5a and 6.5b, respectively. We examined three variants of direct modification: naive (i.e., $\mathtt{qr}\{\boldsymbol{RP}\}$); our optimized algorithm; and our optimized algorithm, while allowing parallel modification of independent blocks. We also examined four variants of modification via re-factorization (visible in Fig. 6.4): naive, i.e., complete re-factorization; incremental re-factorization, matching iSAM2; and our optimized algorithm, with and without parallelization. For fairness, our implementation of all variants relied on the standard `qr` function in MATLAB. To attain unbiased results, we generated 100 random pose permutations, to be used for comparison. In order to still keep the number of blocks consistent, when generating each permutation, we first randomly selected two blocks of variables, each with a random number of poses, and within each of those, generated a random pose permutation.

The average results of applying these permutations, for each of the compared algorithms, are summarized in Table 6.1; the table presents the execution time, and the number of NZs (NNZ), indicating the added fill-in due to numerical errors. The sparsity pattern of the modified factor $\boldsymbol{R_p}$ (considering a typical permutation) can be seen in Fig. 6.5c and Fig. 6.5d. Several observations can be made. First, the NNZ within each algorithm category is consistent, though direct modification generally yields additional fill-in in comparison to re-factorization. Nonetheless, in terms of execution time, direct optimization beats re-factorization by a large margin. As we expected, even without parallelization, in each category, our optimized algorithm achieves significantly better performance in comparison to the baseline methods. Allowing parallel computing can obviously further improve the performance, in direct relation to the number of concurrently modified blocks.

Table 6.1: Comparison of modification algorithms.

Category	*Algorithm*	*Runtime (ms)*	*NNZ*
Direct modification	Naive	87	390k
	Optimized	13	392k
	Optimized (parallel)	11	392k
Re-factorization	Naive	336	304k
	Incremental (iSAM2)	229	304k
	Optimized	83	307k
	Optimized (parallel)	70	307k

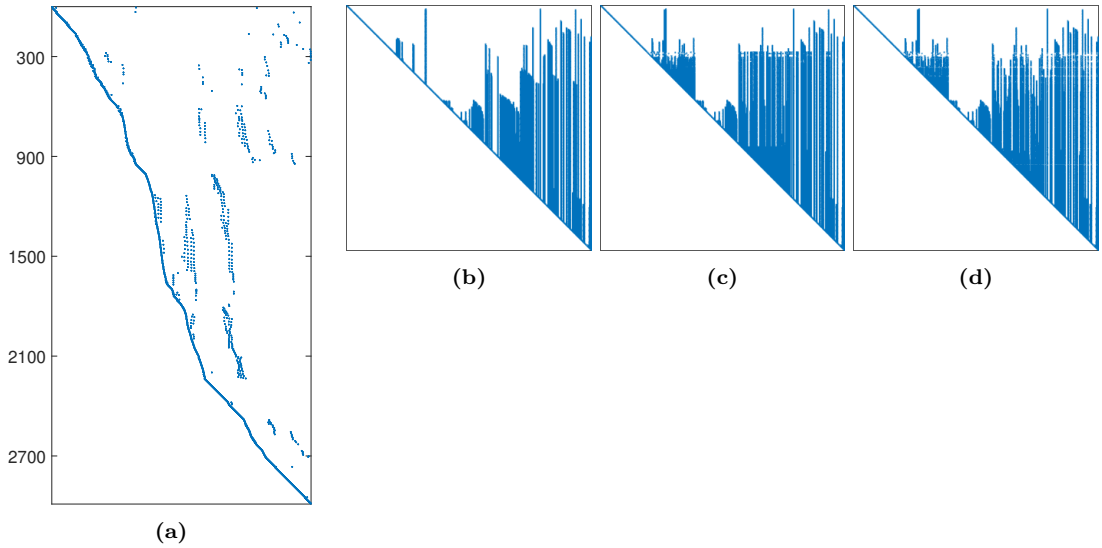


Figure 6.5: The sparsity pattern of the coefficient matrix $\boldsymbol{J}$ (a), its upper triangular factor $\boldsymbol{R}$ (b), and the modified factor $\boldsymbol{R_p}$, calculated using the two variants our optimized modification algorithm: direct modification (c), and re-factorization (d).

Chapter 7

Conclusion

In an attempt to allow efficient solution of decision problems, and, specifically, high-dimensional belief space planning (BSP) problems, we introduced a new conceptual solution paradigm, via conscious simplification of the problem. To conclude, let us summarize our contributions, and discuss some of the potential extensions of this work.

Theoretical Framework We started by suggesting a general theoretical framework for formulation of optimality guarantees for simplified decision making. To provide such guarantees, we presented a novel analysis approach, by measuring the simplification offset, which represents the size of (confidence) intervals for the candidates' values. We explained how to translate these intervals to two types of loss guarantees: pre-solution and post-solution guarantees. We explained how we can optimally utilize intervals which are deterministic, probabilistic, asymmetric, and even of different sizes. We also presented the idea of action consistency, and explained how to mitigate simplification bias. This analysis approach also allows us to identify sub-optimal actions, based on a simplified solution, and perform fast action elimination; this is particularly useful when considering sequential refinement of the solution. Since the framework separates interval derivation from the guarantee formulation, we can potentially use this framework to examine various problem domains and simplifications.

Belief Sparsification We then addressed the challenging problem of decision making in the belief space. We recognized that examining beliefs over high-dimensional states ("smoothing") can help us to improve accuracy and generality; yet, this paradigm makes the problem extremely challenging, due to the "curse of dimesionality", which is exponentially amplified during planning. Practically, we claimed that for identification of the best candidate action, we can utilize a sparse approximation of the initial belief, and provided a scalable algorithm for generation of such sparse approximations. This versatile algorithm can generate approximations of different degrees, based on the subset of state variables selected for the sparsification. Using our theoretical framework, we identified that by considering the problem's *uninvolved variables*, we can provide an approximation which is *guaranteed* to preserve the action selection, and induces no loss. In any case, this approximation does not compromise the accuracy of the state inference process, as, after selecting an action, it should be applied on the *original* belief, which remains exact.

PIVOT We followed by presenting another novel idea to improve the efficiency of planning in the belief space, with no sacrifice in accuracy. First, we recognized that the order of variables in the current belief, as expressed in its "square root matrix" $\boldsymbol{R}$ or Bayes net, determines which

subset of them would be affected by an upcoming update. We also recognized that during planning, to evaluate the different candidates, we should perform multiple such updates on the same belief, "in parallel". Thus, when facing such a scenario, we suggested to perform a precursory optimization to the variable order (i.e., belief representation), in order to reduce the total computational cost of the following updates. We referred to this reordering paradigm as PIVOT: Predictive Incremental Variable Ordering Tactic, and presented two of its variants: the basic and inherently incremental variant, which is easily derived by identifying the involvement classes of the state variables; and an optimized variant, which also takes into consideration fill-reduction during variable classification and ordering. Essentially, these orders help reducing the update cost by bringing forwards "loop closing variables" in the variable order, and sparing the update of "uninvolved variables", as identified by the candidate updates.

We also saw that as a "by-product" of applying the tactic, we are able to improve the efficiency of the state inference process: if we maintain the PIVOT order after the planning session, and our prediction is in-line with the ground truth during execution, then we would similarly reduce the cost of loop closures, when they actually occur. This benefit is derived from a wider view of the system, which considers both the state inference, and planning processes. If we were to examine only the inference process, there would be no point in actively altering the variable order in the initial belief, as the cost of such reordering would be equivalent to actually performing the original update.

Efficient Square Root Modification To support PIVOT, we addressed the problem of modifying the upper triangular matrix $\boldsymbol{R}$, to convey this reordering of the variables. We identified three main contributions, to allow efficient computation. First, variable reordering has only a local effect on the factor; meaning, $\boldsymbol{R}$ is affected only between its rows which correspond to the first and last permuted variables. Second, this row block can be divided into independent sub-blocks, which can be modified independently, and even in parallel. Third, to modify each of these blocks, it is sufficient to examine only the square portion of it, around the matrix diagonal, in order to derive the transformation required for the modification. We utilized these conclusions in a novel modification algorithm, and examined two variants of it: directly modifying $\boldsymbol{R}$, and combining (partial) re-factorization. As our simulation proves, the first enjoys lower computation time, while the latter enjoys higher accuracy. Nonetheless, considering either variant, our optimized algorithm significantly improves the performance over state-of-the-art. Overall, as the two variants show superiority in different criteria, selecting the one to use in a matter of preference.

Demonstration To prove the benefits of our ideas, we applied them in several experiments. We started by utilizing belief sparsification in a highly realistic active SLAM simulation. There, we showed that using sparsification of the uninvolved variables, planning time can be significantly reduced, while, as mentioned, guaranteeing no loss in the quality of solution. We then showed that planning time can be reduced even further, if using full sparsification (i.e., sparsifying all the state variables); in practice, for this configuration, we experienced no loss in the quality of solution, as well. Nonetheless, we demonstrated how, using our theoretical framework, the potential loss can be bounded. We further demonstrated the utilization of our theoretical framework, by performing action elimination, based on sparsified beliefs, in a sensor placement simulation. We even demonstrated (in Appendix A) how to derive the (probabilistic) offset for a sampling-based simplification, and showed that it was only dependent on the *number* of samples, and can hence be derived pre-solution.

Finally, we demonstrated the effectiveness of PIVOT, by applying it to the same active-SLAM simulation. There, we solved the sequence of planning and state inference sessions, using various variable ordering tactics. We proved that, using PIVOT, we were able to significantly reduce the

computation time of both the planning and inference, with only a slight increase in fill-in (the memory footprint), and with no sacrifice in accuracy. In that experiment, we also utilized our square root modification algorithm, to practically apply the PIVOT order.

Future Work The proposed paradigm offers many possible future research directions. In general, other belief simplification methods, besides the provided sparsification algorithm, can be used in similar ways. We can also examine other simplification methods, such as altering the action set or the objective function. Developing simplification methods for more general beliefs, such as multi-modal Gaussians, can hold important practical significance. In any case, the impact of all these methods on the action selection and the potential loss can be examined using our framework. In fact, we can use the framework as a basis to design simplification methods. For example, we can start by asserting a constraint on the offset, and then attempt to reverse-engineer the expression, in order to understand in what manner the belief can be simplified, without violation. It would also be valuable to utilize the framework to quantify the loss in optimality due to model uncertainty.

In the context of PIVOT, since it does not make any assumptions on the belief structure, it would be interesting to verify its benefits when applied to other belief topologies, such as in full-SLAM (vs. pose-SLAM), or when considering non-binary factors. We may also be able to further optimize the order, by utilizing more advanced fill-reducing concepts, such as graph dissection (Krauthausen et al., 2006), or the Bayes tree (Kaess et al., 2010). These concepts can also be utilized to improve our square root modification algorithm, if we can first generalize it to support beliefs which are represented with graphical models (and not only matrices). Finally, we may also be able to examine the applicability of PIVOT to other contexts, such as multi-hypothesis or multi-modal beliefs (Pathak et al., 2018), where we might also face branching belief updates.

Overall, with the versatility of these ideas, we hope this work will prove to yield a significant contribution to the research community.

Bibliography

Abel, D., Hershkowitz, D. E., Barth-Maron, G., Brawner, S., O'Farrell, K., MacGlashan, J. and Tellex, S. (2015), Goal-based action priors., *in* 'ICAPS', pp. 306–314.

Agarwal, P. and Olson, E. (2012), Variable reordering strategies for slam, *in* 'IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)', IEEE, pp. 3844–3850.

Agha-mohammadi, A.-a., Agarwal, S., Kim, S.-K., Chakravorty, S. and Amato, N. M. (2018), 'Slap: Simultaneous localization and planning under uncertainty via dynamic replanning in belief space', *IEEE Trans. Robotics* **34**(5), 1195–1214.

Agha-Mohammadi, A.-A., Chakravorty, S. and Amato, N. M. (2014), 'FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements', *Intl. J. of Robotics Research* **33**(2), 268–304.

Agullo, E., Buttari, A., Guermouche, A. and Lopez, F. (2013), Multifrontal qr factorization for multicore architectures over runtime systems, *in* 'European Conf. on Parallel Processing', Springer, pp. 521–532.

Anderson, T. W. (1969), Confidence limits for the expected value of an arbitrary bounded random variable with a continuous distribution function, Technical report, Department of statistics, Stanford University.

Bahadur, R. R. and Savage, L. J. (1956), 'The nonexistence of certain statistical procedures in nonparametric problems', *The Annals of Mathematical Statistics* **27**(4), 1115–1122.

Bellman, R., Corporation, R. and Collection, K. M. R. (1957), *Dynamic Programming*, Rand Corporation research study, Princeton University Press.

Ben-Gal, I. (2008), 'Bayesian networks', *Encyclopedia of statistics in quality and reliability* **1**.

Besl, P. and McKay, N. (1992), 'A method for registration of 3-D shapes', *IEEE Trans. Pattern Anal. Machine Intell.* **14**(2).

Bonet, B. and Geffner, H. (2000), Planning with incomplete information as heuristic search in belief space, *in* 'Intl. Conf. on Artificial Intelligence Planning Systems', AAAI Press, pp. 52–61.

Bopardikar, S. D., Englot, B., Speranzon, A. and van den Berg, J. (2016), 'Robust belief space planning under intermittent sensing via a maximum eigenvalue-based bound', *IJRR* **35**(13), 1609–1626.

Boucheron, S., Lugosi, G. and Massart, P. (2013), *Concentration inequalities: A nonasymptotic theory of independence*, Oxford university press.

Boyen, X. and Koller, D. (1998), Tractable inference for complex stochastic processes, *in* 'Proc. 14^{th} Conf. on Uncertainty in AI (UAI)', Madison, WI, pp. 33–42.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. D. and Leonard, J. J. (2016), 'Simultaneous localization and mapping: Present, future, and the robust-perception age', *IEEE Trans. Robotics* **32**(6), 1309 – 1332.

Carlevaris-Bianco, N. and Eustice, R. M. (2014), Conservative edge sparsification for graph slam node removal, *in* 'IEEE Intl. Conf. on Robotics and Automation (ICRA)', pp. 854–860.

Carlevaris-Bianco, N., Kaess, M. and Eustice, R. M. (2014), 'Generic node removal for factor-graph SLAM', *IEEE Trans. Robotics* **30**(6), 1371–1385.

Chaves, S. M. and Eustice, R. M. (2016), Efficient planning with the Bayes tree for active SLAM, *in* 'Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on', IEEE, pp. 4664–4671.

Chen, Y., Davis, T. A., Hager, W. W. and Rajamanickam, S. (2008), 'Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate', *ACM Transactions on Mathematical Software (TOMS)* **35**(3), 1–14.

Coulter, R. C. (1992), Implementation of the pure pursuit path tracking algorithm, Technical report, Robotics Institute, Carnegie-Mellon University.

Davis, T. A. (2006), *Direct Methods for Sparse Linear Systems*, Fundamentals of Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, United States.

Davis, T., Gilbert, J., Larimore, S. and Ng, E. (2004), 'A column approximate minimum degree ordering algorithm', *ACM Trans. Math. Softw.* **30**(3), 353–376.

Davis, T. and Hager, W. (1996), 'Modifying a sparse Cholesky factorization', *SIAM Journal on Matrix Analysis and Applications* **20**(3), 606–627.

Dellaert, F. (2012), Factor graphs and GTSAM: A hands-on introduction, Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology.

Dellaert, F. and Kaess, M. (2006), 'Square Root SAM: Simultaneous localization and mapping via square root information smoothing', *Intl. J. of Robotics Research* **25**(12), 1181–1203.

Dellaert, F. and Kaess, M. (2017), 'Factor graphs for robot perception', *Foundations and Trends in Robotics* **6**(1-2), 1–139.

Dellaert, F., Kipp, A. and Krauthausen, P. (2005), A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping, *in* 'Proc. 22^{nd} AAAI National Conference on AI', Pittsburgh, PA, pp. 1261–1266.

Even-Dar, E., Mannor, S. and Mansour, Y. (2006), 'Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems', *Journal of machine learning research* **7**(Jun), 1079–1105.

Farhi, E. I. and Indelman, V. (2017), Towards efficient inference update through planning via jip - joint inference and belief space planning, *in* 'IEEE Intl. Conf. on Robotics and Automation (ICRA)'.

Fréchet, M. (1935), 'Généralisation du théoreme des probabilités totales', *Fundamenta mathematicae* **1**(25), 379–387.

Gilbert, J. R. and Tarjan, R. E. (1986), 'The analysis of a nested dissection algorithm', *Numerische mathematik* **50**(4), 377–404.

Gill, P., Golub, G., Murray, W. and Saunders, M. (1974), 'Methods for modifying matrix factorizations', *Mathematics and Computation* **28**(126), 505–535.

Golub, G. and Loan, C. V. (1996), *Matrix Computations*, third edn, Johns Hopkins University Press, Baltimore.

Hämmerlin, G. and Hoffmann, K.-H. (2012), *Numerical mathematics*, Springer Science & Business Media.

Harville, D. A. (1998), 'Matrix algebra from a statistician's perspective', *Technometrics* **40**(2), 164–164.

Hoeffding, W. (1994), Probability inequalities for sums of bounded random variables, *in* 'The Collected Works of Wassily Hoeffding', Springer, pp. 409–426.

Hsiung, J., Hsiao, M., Westman, E., Valencia, R. and Kaess, M. (2018), Information sparsification in visual-inertial odometry, *in* 'IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)', pp. 1146–1153.

Huang, G., Kaess, M. and Leonard, J. (2012), Consistent sparsification for graph optimization, *in* 'Proc. of the European Conference on Mobile Robots (ECMR)', pp. 150 – 157.

Ila, V., Polok, L., Solony, M. and Svoboda, P. (2017), 'SLAM++ - a highly efficient and temporally scalable incremental slam framework', *Intl. J. of Robotics Research* **36**(2), 210–230.

Indelman, V. (2015), Towards information-theoretic decision making in a conservative information space, *in* 'American Control Conference', pp. 2420–2426.

Indelman, V. (2016), 'No correlations involved: Decision making under uncertainty in a conservative sparse information space', *IEEE Robotics and Automation Letters (RA-L)* **1**(1), 407–414.

Indelman, V., Carlone, L. and Dellaert, F. (2015), 'Planning in the continuous domain: a generalized belief space approach for autonomous navigation in unknown environments', *Intl. J. of Robotics Research* **34**(7), 849–882.

Kaelbling, L. P., Littman, M. L. and Cassandra, A. R. (1998), 'Planning and acting in partially observable stochastic domains', *Artificial intelligence* **101**(1), 99–134.

Kaess, M., Ila, V., Roberts, R. and Dellaert, F. (2010), The Bayes tree: Enabling incremental reordering and fluid relinearization for online mapping, Technical Report MIT-CSAIL-TR-2010-021, Computer Science and Artificial Intelligence Laboratory, MIT.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. and Dellaert, F. (2012), 'iSAM2: Incremental smoothing and mapping using the Bayes tree', *Intl. J. of Robotics Research* **31**(2), 217–236.

Kaess, M., Ranganathan, A. and Dellaert, F. (2008), 'iSAM: Incremental smoothing and mapping', *IEEE Trans. Robotics* **24**(6), 1365–1378.

Kakade, S. M. (2003), On the sample complexity of reinforcement learning, PhD thesis, University College London.

Karaman, S. and Frazzoli, E. (2011), 'Sampling-based algorithms for optimal motion planning', *Intl. J. of Robotics Research* **30**(7), 846–894.

Kavraki, L., Svestka, P., Latombe, J.-C. and Overmars, M. (1996), 'Probabilistic roadmaps for path planning in high-dimensional configuration spaces', *IEEE Trans. Robot. Automat.* **12**(4), 566–580.

Kendall, M. G. (1948), *Rank Correlation Methods*, Griffin.

Kepner, J. and Gilbert, J. (2011), *Graph algorithms in the language of linear algebra*, SIAM.

Khosoussi, K., Giamou, M., Sukhatme, G. S., Huang, S., Dissanayake, G. and How, J. P. (2018), 'Reliable graph topologies for SLAM', *Intl. J. of Robotics Research* .

Kim, A. and Eustice, R. M. (2014), 'Active visual SLAM for robotic area coverage: Theory and experiment', *Intl. J. of Robotics Research* **34**(4-5), 457–475.

Kitanov, A. and Indelman, V. (2019), 'Topological information-theoretic belief space planning with optimality guarantees', *arXiv preprint arXiv:1903.00927* .

Kochenderfer, M. J. (2015), *Decision making under uncertainty: theory and application*, MIT press.

Koenig, N. and Howard, A. (2004), Design and use paradigms for gazebo, an open-source multi-robot simulator, *in* 'IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)'.

Koller, D. and Friedman, N. (2009), *Probabilistic Graphical Models: Principles and Techniques*, The MIT Press.

Kopitkov, D. and Indelman, V. (2017), 'No belief propagation required: Belief space planning in high-dimensional state spaces via factor graphs, matrix determinant lemma and re-use of calculation', *Intl. J. of Robotics Research* **36**(10), 1088–1130.

Kopitkov, D. and Indelman, V. (2019), 'General purpose incremental covariance update and efficient belief space planning via factor-graph propagation action tree', *Intl. J. of Robotics Research* .

Krauthausen, P., Dellaert, F. and Kipp, A. (2006), Exploiting locality by nested dissection for square root smoothing and mapping, *in* 'Robotics: Science and Systems (RSS)'.

Kretzschmar, H. and Stachniss, C. (2012), 'Information-theoretic compression of pose graphs for laser-based SLAM', *Intl. J. of Robotics Research* **31**(11), 1219–1230.

Künzi, H.-P. A. (2001), Nonsymmetric distances and their associated topologies: about the origins of basic ideas in the area of asymmetric topology, *in* 'Handbook of the history of general topology', Springer, pp. 853–968.

LaValle, S. M. (2006), *Planning algorithms*, Cambridge university press.

Mannor, S. and Tsitsiklis, J. N. (2004), 'The sample complexity of exploration in the multi-armed bandit problem', *Journal of Machine Learning Research* **5**(Jun), 623–648.

Manski, C. F. (1988), 'Ordinal utility models of decision making under uncertainty', *Theory and Decision* **25**(1), 79–104.

McAllester, D. A. and Singh, S. (1999), Approximate planning for factored pomdps using belief state simplification, *in* 'UAI', Morgan Kaufmann Publishers Inc., pp. 409–416.

Mu, B., Paull, L., Agha-Mohammadi, A.-A., Leonard, J. J. and How, J. P. (2017), 'Two-stage focused inference for resource-constrained minimal collision navigation', *IEEE Trans. Robotics* **33**(1), 124–140.

Nakhost, H. and Müller, M. (2010), Action elimination and plan neighborhood graph search: Two algorithms for plan improvement., *in* 'ICAPS', pp. 121–128.

Pathak, S., Thomas, A. and Indelman, V. (2018), 'A unified framework for data association aware robust belief space planning and perception', *Intl. J. of Robotics Research* **32**(2-3), 287–315.

Patil, S., Kahn, G., Laskey, M., Schulman, J., Goldberg, K. and Abbeel, P. (2014), Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation, *in* 'Intl. Workshop on the Algorithmic Foundations of Robotics (WAFR)', pp. 515–533.

Pineau, J., Gordon, G. J. and Thrun, S. (2006), 'Anytime point-based approximations for large POMDPs.', *J. of Artificial Intelligence Research* **27**, 335–380.

Platt, R., Tedrake, R., Kaelbling, L. and Lozano-Pérez, T. (2010), Belief space planning assuming maximum likelihood observations, *in* 'Robotics: Science and Systems (RSS)', Zaragoza, Spain, pp. 587–593.

Porta, J. M., Vlassis, N., Spaan, M. T. and Poupart, P. (2006), 'Point-based value iteration for continuous pomdps', *J. of Machine Learning Research* **7**, 2329–2367.

Prentice, S. and Roy, N. (2009), 'The belief roadmap: Efficient planning in belief space by factoring the covariance', *Intl. J. of Robotics Research* **28**(11-12), 1448–1465.

Rosman, B. and Ramamoorthy, S. (2012), What good are actions? accelerating learning using learned action priors, *in* 'Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on', IEEE, pp. 1–6.

Ross, S., Pineau, J., Paquet, S. and Chaib-Draa, B. (2008), 'Online planning algorithms for pomdps', *J. of Artificial Intelligence Research* **32**, 663–704.

Roy, N., Gordon, G. J. and Thrun, S. (2005), 'Finding approximate pomdp solutions through belief compression', *J. Artif. Intell. Res.(JAIR)* **23**, 1–40.

Sherstov, A. and Stone, P. (2005), Action-space knowledge transfer in mdps: Formalism, suboptimality bounds, and algorithms, *in* 'Conference on Learning Theory'.

Silver, D. and Veness, J. (2010), Monte-carlo planning in large pomdps, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 2164–2172.

Sim, R., Elinas, P., Griffin, M. and Little, J. (2005), Vision-based SLAM using the Rao-Blackwellised particle filter, *in* 'Proc. of the IJCAI Workshop on Reasoning with Uncertainty in Robotics (RUR)'.

Stachniss, C., Grisetti, G. and Burgard, W. (2005), Information gain-based exploration using Rao-Blackwellized particle filters, *in* 'Robotics: Science and Systems (RSS)', pp. 65–72.

Stachniss, C., Haehnel, D. and Burgard, W. (2004), Exploration with active loop-closing for FastSLAM, *in* 'IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)'.

Strehl, A. L. (2007), Probably approximately correct (PAC) exploration in reinforcement learning, PhD thesis, Rutgers University.

Strehl, A. L., Li, L. and Littman, M. L. (2009), 'Reinforcement learning in finite mdps: Pac analysis.', *Journal of Machine Learning Research* **10**(11).

Thrun, S. (2000), Monte carlo pomdps, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 1064–1070.

Thrun, S., Burgard, W. and Fox, D. (2005), *Probabilistic Robotics*, The MIT press, Cambridge, MA.

Thrun, S., Liu, Y., Koller, D., Ng, A., Ghahramani, Z. and Durrant-Whyte, H. (2004), 'Simultaneous localization and mapping with sparse extended information filters', *Intl. J. of Robotics Research* **23**(7-8), 693–716.

Touchette, S., Gueaieb, W. and Lanteigne, E. (2016), 'Efficient cholesky factor recovery for column reordering in simultaneous localisation and mapping', *J. of Intelligent and Robotic Systems* **84**(1-4), 859–875.

Van Den Berg, J., Patil, S. and Alterovitz, R. (2012), 'Motion planning under uncertainty using iterative local optimization in belief space', *Intl. J. of Robotics Research* **31**(11), 1263–1278.

Voss, C., Moll, M. and Kavraki, L. E. (2015), A heuristic approach to finding diverse short paths, *in* 'IEEE Intl. Conf. on Robotics and Automation (ICRA)', pp. 4173–4179.

Yannakakis, M. (1981), 'Computing the minimum fill-in is np-complete', *SIAM Journal on Algebraic Discrete Methods* **2**(1), 77–79.

Ye, N., Somani, A., Hsu, D. and Lee, W. S. (2017), 'Despot: Online pomdp planning with regularization', *JAIR* **58**, 231–266.

Yedidia, J. S., Freeman, W. T. and Weiss, Y. (2003), 'Understanding belief propagation and its generalizations', *Exploring artificial intelligence in the new millennium* **8**, 236–239.

Appendices

Appendix A

Simplified Decision Making: Probabilistic Simplifications

A.1 Analyzing Probabilistic Asymmetric Simplifications

A.1.1 Probabilistic Simplification Loss: Formalizing Near-Optimality

We now wish to extend our theoretical framework which we presented in Chapter 2, to allow derivation of probabilistic simplification guarantees. Such guarantees are especially usable when analyzing simplification methods which have a stochastic component; e.g., we may consider the objective function V (3.27), which we previously defined for planning in the belief space, and which often *must* be approximated, via sub-sampling of random hypotheses. For such simplifications, it is often impossible to provide "absolute guarantees", as considered before, and we thus prefer to seek more lenient guarantees, which hold with a certain confidence level. We begin by defining:

Definition 7. Consider a decision problem $\mathcal{P}$ (as previously defined), and a candidate action a (*a hypothesis*). The candidate is (ϵ, α)-near optimal, with confidence level $1-\alpha \in [0,1]$ and loss limit $\epsilon \in [0, \infty]$, if it satisfies

$$\mathbb{P}(\mathit{loss} \leq \epsilon) \geq 1-\alpha, \text{ where } \mathit{loss} \doteq V(\boldsymbol{\xi}, a^*) - V(\boldsymbol{\xi}, a), \tag{A.1}$$

where $a^* \doteq \text{argmax}_{a \in \mathcal{A}} V(\boldsymbol{\xi}, a)$ is the optimal candidate.

In our context, we typically consider the hypothesis to be returned as the solution of a simplified decision problem $\mathcal{P}_s$, i.e., the candidate which yielded the highest approximated value $V_s(\boldsymbol{\xi}_s, a)$. The definition can thus be viewed as a generalization of the simplification loss (Definition 2), which now considers the loss to be a random variable, and requires it to be in the (confidence) interval $[0, \epsilon]$, with confidence $1-\alpha$.

A.1.2 Probabilistic Simplification Offset

Similarly to the loss, we can also extend the notion of simplification offset (Definition 3), by allowing it to be probabilistic, and match the structure of a confidence interval.

Definition 8. Considering a confidence level $1-\alpha \in [0,1]$, the $(1-\alpha)$-*confidence simplification offset* of a candidate $a \in \mathcal{A}$ is

$$\delta_\alpha(\mathcal{P}, \mathcal{P}_s, a) \doteq \min\{\epsilon \in \mathbb{R}^+ \mid \mathbb{P}(|V(a) - V_s(a)| \leq \epsilon) \geq 1-\alpha\}. \tag{A.2}$$

Overall, the $(1-\alpha)$-*confidence simplification offset* between $\mathcal{P}$ and $\mathcal{P}_s$ is

$$\Delta_\alpha(\mathcal{P}, \mathcal{P}_s) \doteq \max_{a\in\mathcal{A}}\{\delta_\alpha(\mathcal{P}, \mathcal{P}_s, a)\} \tag{A.3}$$

Meaning, $\delta_\alpha(V, V_s, a)$ is the minimal limit ϵ, for which we can guarantee with at least $1-\alpha$ confidence, that the offset of a does not exceed. When $\alpha = 0$, this definition reduces back to the original deterministic offset. Again, we may extend this notion, by measuring the lower and upper offsets separately:

Definition 9. Considering confidence levels $1-\alpha_1$, $1-\alpha_2 \in [0,1]$, the $(1-\alpha_1, 1-\alpha_2)$-*confidence simplification offset* of a candidate $a \in \mathcal{A}$ is

$$\delta_{(\alpha_1,\alpha_2)}(\mathcal{P}, \mathcal{P}_s, a) \doteq (\underline{\delta}_{\alpha_1}(\mathcal{P}, \mathcal{P}_s, a), \overline{\delta}_{\alpha_2}(\mathcal{P}, \mathcal{P}_s, a)), \tag{A.4}$$

where

$$\underline{\delta}_{\alpha_1}(\mathcal{P}, \mathcal{P}_s, a) \doteq \min\{\epsilon_1 \in \mathbb{R}^+ \mid \mathbb{P}(V(a) - V_s(a) \geq -\epsilon_1) \geq 1-\alpha_1,\} \tag{A.5}$$

$$\overline{\delta}_{\alpha_2}(\mathcal{P}, \mathcal{P}_s, a) \doteq \min\{\epsilon_2 \in \mathbb{R}^+ \mid \mathbb{P}(V(a) - V_s(a) \leq \quad \epsilon_2) \geq 1-\alpha_2.\} \tag{A.6}$$

Overall, the $(1-\alpha_1, 1-\alpha_2)$-*confidence simplification offset* between $\mathcal{P}$ and $\mathcal{P}_s$ is

$$\Delta_{(\alpha_1,\alpha_2)}(\mathcal{P}, \mathcal{P}_s) \doteq (\underline{\Delta}_{\alpha_1}(\mathcal{P}, \mathcal{P}_s), \overline{\Delta}_{\alpha_2}(\mathcal{P}, \mathcal{P}_s)), \tag{A.7}$$

where

$$\underline{\Delta}_{\alpha_1}(\mathcal{P}, \mathcal{P}_s) \doteq \max_{a\in\mathcal{A}}\{\underline{\delta}_{\alpha_1}(\mathcal{P}, \mathcal{P}_s, a)\}, \tag{A.8}$$

$$\overline{\Delta}_{\alpha_2}(\mathcal{P}, \mathcal{P}_s) \doteq \max_{a\in\mathcal{A}}\{\overline{\delta}_{\alpha_2}(\mathcal{P}, \mathcal{P}_s, a)\}. \tag{A.9}$$

According to Fréchet inequalities Fréchet (1935), we know that the intersection of any two events A and B, satisfies

$$\mathbb{P}(A \cap B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cup B) \geq \mathbb{P}(A) + \mathbb{P}(B) - 1. \tag{A.10}$$

Thus, for every $a \in \mathcal{A}$, the probability (confidence) for the upper and lower offsets – $\underline{\delta}_{\alpha_1}(a)$ and $\overline{\delta}_{\alpha_2}(a)$ – to hold simultaneously is $\geq 1-(\alpha_1+\alpha_2)$. When this occurs, the combination of the two events conveys a(n asymmetric) confidence interval for the real value $V(a)$, around the respective simplified value $V_s(a)$, such that

$$\mathbb{P}\Big(V_s(a) - \underline{\delta}_{\alpha_1}(a) \leq V(a) \leq V_s(a) + \overline{\delta}_{\alpha_2}(a)\Big) \geq 1-(\alpha_1+\alpha_2). \tag{A.11}$$

For each a, this interval over $V(a)$ is contained between the Upper Confidence Bound (UCB), and the Lower Confidence Bound (LCB). As mentioned, the offset represents only the *size* of this interval, and not its *location* on the value axis (which is around $V_s(a)$).

A.2 From Offset to Loss Guarantees & Action Elimination

A.2.1 Post-Solution Loss Guarantees

Again, let us first focus on deriving "post-solution" guarantees. Thus, assume we have indeed solved the simplified problem $\mathcal{P}_s$ (i.e., calculated $V_s(a)$, $\forall a \in \mathcal{A}$).

As we recall, in our context, the loss is defined as: $V(a^*) - V(a_s^*)$, where a_s^* is the selected candidate. Surely, we do not have access to the real values, but let us consider that for each $a \in \mathcal{A}$, we know the $(1-\beta_a, 1-\gamma_a)$-confidence offset, $\delta_{(\beta_a,\gamma_a)}(a)$, and, by such, a confidence interval for $V(a)$ (as defined in (A.11)). Given this knowledge, which we later clarify how to practically reach, we wish to provide guarantees for the loss of the solution.

By definition, the "highest" interval, i.e., the one located around the maximal $V_s(a)$, corresponds to the selected candidate a_s^* (otherwise, it would not have been selected). Let us mark the respective offset of this candidate as $\delta_{(\beta_s^*,\gamma_s^*)}(a_s^*)$.

Now, let us assume the real optimal candidate a^* is not actually the selected candidate a_s^*, but another candidate a. From the definition of the offset we know that

$$\mathbb{P}(V(a_s^*) \geq V_s(a_s^*) - \underline{\delta}_{\beta_s^*}(a_s^*)) \qquad \geq 1-\beta_s^* \tag{A.12}$$

$$\mathbb{P}(V(a) \leq V_s(a) + \overline{\delta}_{\gamma_a}(a)) \qquad \geq 1-\gamma_a \tag{A.13}$$

Note that unlike before, these are the upper and lower bounds of *different* candidates. The probability (confidence) for the two events examined in (A.12)-(A.13) to happen simultaneously is $\geq 1-(\beta_s^*+\gamma_a)$. When this occurs, we can infer that:

$$\mathit{loss}(\mathcal{P},\mathcal{P}_s) \leq \max\{\epsilon_a, 0\}, \qquad \text{where} \qquad \epsilon_a \doteq \underline{\delta}_{\beta_s^*}(a_s^*) + \overline{\delta}_{\gamma_a}(a) - \big(V_s(a_s^*) - V_s(a)\big). \tag{A.14}$$

with at least $1-\min\{\alpha_a, 1\}$ confidence, where $\alpha_a \doteq \beta_s^* + \gamma_a$. Note that we limited the loss and confidence levels to their correct range: α_a must be ≤ 1, and ϵ_a must be ≥ 0.

As we recall, the loss limit ϵ_a, is determined by the overlap of the confidence interval of a, with the interval of a_s^*. If the pair of intervals do not overlap, i.e., the upper bound for the value of a is lower than the lower bound for the value of a_s^*, i.e.,

$$V_s(a_s^*) + \underline{\delta}_{\beta_s^*}(a_s^*) > V_s(a) + \overline{\delta}_{\gamma_a}(a), \tag{A.15}$$

then it implies that ϵ_a should be negative. Thus, a could not actually have been the real optimal action in the first place, and we can infer that this candidate is *probably sub-optimal* (with $1-\alpha_a$ confidence), and can *eliminate* this candidate.

Surely, we do not actually know which candidate is the real optimal candidate a^*. We should thus perform this interval overlap analysis for each candidate $a \neq a_s^*$, assuming it is the optimal candidate. If there are remaining, un-eliminated candidates (besides a_s^*), we may return only the most conservative guarantee – that the solution is $(\max\{\epsilon,0\}, \min\{\alpha,1\})$-near optimal, where

$$\epsilon = \max_{a \neq a_s^*}\{\epsilon_a\}, \tag{A.16}$$

$$\alpha = \max_{a \neq a_s^*}\{\alpha_a\}. \tag{A.17}$$

A.2.2 Optimized Guarantees: Balancing α and ϵ

The overlap of each pair of intervals is determined by their location (on the value axis) and their size. As mentioned, their location is set around the respective approximated values $V_s(a)$, which we consider here to be set; however, their size, which is represented with the offset, may vary.

In the previous discussion, we implicitly assumed that we only had access to a single confidence interval for each candidate (with specific size and confidence level). However, from the nature of confidence interval derivation (as we shall see), it is likely that we have a "span" of intervals per candidate, matching a range of confidence levels. For example, we are likely to know $\overline{\delta}_{\gamma_a}(a)$, as a function of γ_a, for every $\gamma_a \in [0,1]$. There is a positive correlation between the interval size and confidence level; i.e., increasing the confidence level increases the size of the interval, and vice versa. This arises a question: when performing the interval overlap analysis, which intervals to consider for each candidate, in order to optimize the yielded guarantees? The answer depends on what we care to provide:

Confidence constraint given a confidence level $1-\alpha$, we wish to return the minimal loss limit ϵ (i.e., the tightest loss bound), which we can guarantee not to surpass, with at least $1-\alpha$ confidence. In this case, we wish to find $\forall a \neq a_s^*$:

$$\epsilon_a^\alpha = \min_{\beta,\gamma} \left\{ \underline{\delta}_\beta(a_s^*) + \overline{\delta}_\gamma(a) \right\} - (V_s(a_s^*) - V_s(a)), \tag{A.18}$$

where

$$1-\alpha \leq 1-(\beta+\gamma) \leq 1. \tag{A.19}$$

Of course, when possible, for each a, the minimal overlap will occur on the lower edge of the inequality constraint, i.e., when we require the minimal confidence levels, and hence minimize the size of the intervals. Then, we can infer that the simplified solution is $(\max\{\max_{a\neq a_s^*}\{\epsilon_a^\alpha\}, 0\}, \alpha)$-near optimal.

Loss constraint given an upper limit for the loss ϵ, we wish to return the maximal confidence level $1-\alpha$, with which we can guarantee not to surpass this limit. In this case, we wish to find $\forall a \neq a_s^*$:

$$\alpha_a^\epsilon = \min_{\underline{\delta}_\beta(a_s^*),\, \overline{\delta}_\gamma(a)} \{\beta+\gamma\}, \tag{A.20}$$

where

$$0 \leq \underline{\delta}_\beta(a_s^*) + \overline{\delta}_\gamma(a) - \big(V_s(a_s^*) - V_s(a)\big) \leq \epsilon. \tag{A.21}$$

Of course, when possible, for each a, the maximum confidence will occur on the upper edge of the inequality constraint, i.e., when we maximize the size of these intervals, and hence the confidence we have in them. For example, when asked to return the confidence level of having no loss (i.e., that $\epsilon = 0$), we shall utilize the intervals of maximal size which do not overlap, i.e., pairs of "touching" intervals. Then, we can infer that the simplified solution is $(\epsilon, \min\{\max_{a\neq a_s^*}\{\alpha_a^\epsilon\}, 1\})$-near optimal.

In both options, if in one of these minimizations, we do not have access to intervals which satisfy the constraints, then we cannot provide a loss guarantee with the given requirement (and should try to tighten the offset first).

As demonstrated in Fig. A.1, the purpose of these minimizations is to give the best guarantee, given our knowledge. It is surely possible that the minimum $\alpha_a^\epsilon/\epsilon_a^\alpha$ would be achieved for a pair of asymmetric intervals. Also, it is possible that for each a, the minimum $\alpha_a^\epsilon/\epsilon_a^\alpha$ would rely on a different interval for a_s^* (in a different confidence level). This means that overall, we rely on multiple intervals of a_s^*, to provide the tightest guarantee, and eliminate sub-optimal candidates. Nonetheless, for each minimization, considering *any* pair of confidence intervals which satisfies the constraint would be enough to provide a certain guarantee. Thus, for a possibly simpler guarantee derivation, we can, e.g., assume the intervals are symmetric around V_s and/or of equal size for each action (i.e., that $\beta = \gamma$, $\forall a$); of course, these assumptions may lead to overly

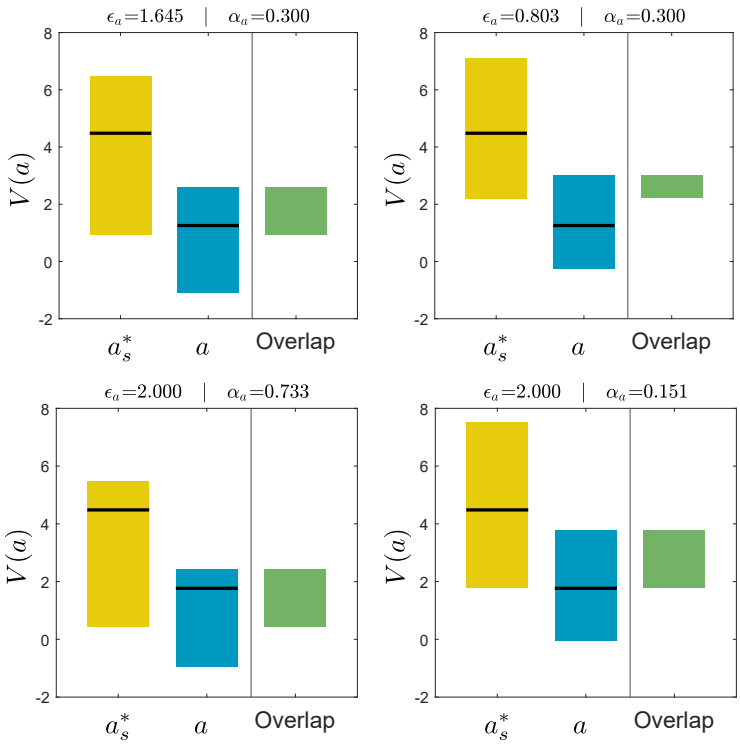


Figure A.1: Derivation of optimized probabilistic loss guarantees via interval overlap analysis. The two top sub-figures demonstrate that two interval pairs, which respect the same confidence constraint, can result in different overlaps (loss bounds). The two bottom pairs respect the same loss (overlap) constraint, but convey guarantees of different confidence levels.

conservative guarantees. When requiring $\epsilon = 0$, for each a, $1-\min\{\alpha_a^0, 1\}$ conveys the confidence level with which we can eliminate a, and $1-\min\{\max_{a\neq a_s^*}\{\alpha_a^0\}, 1\})$ conveys the confidence level with which all actions but a_s^* can be eliminated; this is the probability that the simplified solution is the optimal solution!

A.2.3 Pre-Solution Loss Guarantees

In the previous analysis we explained how to provide guarantees via post-solution analysis. Now, assume instead that we want to perform pre-solution analysis. Thus, let us now assume that we have access to the $(1-\beta_a, 1-\gamma_a)$-confidence offset of each candidate a, but *not* to a calculated solution. Looking back at (A.14), we recall that the upper loss limit ϵ_a conveys the overlap of the confidence intervals of a and a_s^*. However, since we do not have access to the location of the intervals, we have no choice but be conservative, and assume the intervals are located around the same value, in order to consider the maximal potential overlap for each pair of intervals. Meaning, we can (only) infer that

$$\epsilon_a \leq \underline{\delta}_{\beta_s^*}(a_s^*) + \overline{\delta}_{\gamma_a}(a) \tag{A.22}$$

with at least $1-\min\{\alpha_a, 1\}$ confidence, where $\alpha_a \doteq \beta_s^* + \gamma_a$.

From (A.16)-(A.17), we recall that since we did not know which action is actually a^*, in order to provide the loss guarantee, we had to be conservative, and take the maximal ϵ_a and α_a, considering every $a \neq a_s^*$. Now, we also do not know which of the candidates is a_s^*; we thus need to examine the overlap between the intervals of every possible pair of candidates, i.e.,

$$\epsilon_{a',a} = \underline{\delta}_{\beta_{a'}}(a') + \overline{\delta}_{\gamma_a}(a) \tag{A.23}$$

$$\alpha_{a',a} = \beta_{a'} + \gamma_a \tag{A.24}$$

Then, we can conservatively conclude that simplified solution is $(\epsilon, \min\{\alpha, 1\})$-near optimal, where

$$\epsilon = \max_{a'\neq a}\{\epsilon_{a',a}\} \leq \max_{a\in\mathcal{A}}\{\underline{\delta}_{\beta_a}(a)\} + \max_{a\in\mathcal{A}}\{\overline{\delta}_{\gamma_a}(a)\} \tag{A.25}$$

$$\alpha = \max_{a'\neq a}\{\alpha_{a',a}\} \leq \max_{a\in\mathcal{A}}\{\beta_a\} + \max_{a\in\mathcal{A}}\{\gamma_a\} \tag{A.26}$$

If we mark the minimal confidence levels among the lower and upper offsets of every $a \in \mathcal{A}$ with $1-\beta$ and $1-\gamma$ (respectively), then these expressions conveniently reduce to

$$\epsilon \leq \underline{\Delta}_\beta + \overline{\Delta}_\gamma \tag{A.27}$$

$$\alpha \leq \beta + \gamma \tag{A.28}$$

Thus, given $\Delta_{(\beta,\gamma)}$, the $(1-\beta, 1-\gamma)$-confidence offset between $\mathcal{P}$ and $\mathcal{P}_s$, we can guarantee that the simplified solution is $(\underline{\Delta}_\beta + \overline{\Delta}_\gamma, \min\{\beta+\gamma, 1\})$-near optimal. Indeed, since this conclusion depends only on the definition of $\mathcal{P}_s$, and not on the calculated values $V_s(a)$, we can reach this conclusion before calculating the solution. We summarize this claim in the Lemma 9:

Lemma 9. *If*

$$\Delta_{(\alpha_1,\alpha_2)}(\mathcal{P}, \mathcal{P}_s) \leq (\epsilon_1, \epsilon_2), \tag{A.29}$$

then

$$\mathit{loss}(\mathcal{P}, \mathcal{P}_s) \leq \epsilon_1 + \epsilon_2, \tag{A.30}$$

with at least $1-(\alpha_1+\alpha_2)$ *confidence.*

A.3 Practical Application: Empirical Value Estimation

To demonstrate the practical application of this type of probabilistic guarantee derivation, we may examine our previous formulation of decision making in the belief space. There, we encountered the value function V (3.27), which, for an initial belief b_k and candidate policy π, returned the expected accumulated discounted reward ("the expected return"), after applying the policy. We then explained that even though the belief can be propagated parametrically (for each realization of $\mathcal{Z}$), when the observation space is continuous (or just too large), the expectation in V cannot be calculated analytically, but only be empirically estimated via sampling of hypotheses.

Thus, let us mark with V_m a simplified value function, which we define as the empirical mean of the return, using m independently sampled (i.i.d.) hypotheses. We can then use our framework to examine the loss between a decision problem $\mathcal{P} \doteq (b_0, \Pi, V)$ and its simplification $\mathcal{P}_m \doteq (b_0, \Pi, V_m)$. Interestingly, here, we do not actively simplify the problem, as we considered before (via sparsification of the initial belief); instead, the empirical value estimation acts as an "inherent" simplification of an otherwise unattainable original value function. We may nonetheless utilize our framework to quantify its effect on the solution.

A.3.1 Calculating the Offset

For a candidate policy $\pi \in \Pi$, and confidence levels $1-\beta$, $1-\gamma$, we want to derive the offset $\delta_{(\beta,\gamma)}(\mathcal{P}, \mathcal{P}_m, \pi)$. As explained, for each candidate, this offset represents the size of a confidence intervals for the real (unknown) value $V(b_0, \pi)$. We shall start by mentioning that if nothing is known about the distribution of the return, then it is impossible to derive confidence bounds for its mean, due to its sensitivity to the distribution tails (as stated by Bahadur and Savage (1956)). Otherwise, we can generally consider two approaches: (1) deriving parametric confidence intervals, if we know the family of the return distribution; and (2) deriving non-parametric confidence intervals, which rely on other limitations on the distribution.

For conciseness, in the following discussion, we demonstrate how to derive such confidence intervals when the return distribution has a bounded support $[a, b]$. Under this assumption, as shown by Anderson (1969), based on empirical CDF estimation, we can derive confidence intervals for the true mean, such that:

$$\mathbb{P}\Big(V(\pi) - V_m(\pi) \geq -\epsilon' \cdot (b-a)\Big) \geq 1-\alpha, \tag{A.31}$$

$$\mathbb{P}\Big(V(\pi) - V_m(\pi) \leq \epsilon' \cdot (b-a)\Big) \geq 1-\alpha, \tag{A.32}$$

$$\text{where } \epsilon' = \sqrt{\frac{\ln \frac{1}{\alpha}}{2m}}, \ \forall \alpha \in [0,1]. \tag{A.33}$$

From the definition of the offset, we can easily conclude that

$$\overline{\delta}_\alpha(V, V_m, \pi) = \underline{\delta}_\alpha(V, V_m, \pi) = \sqrt{\frac{\ln \frac{1}{\alpha}}{2m}} \cdot (b-a) \tag{A.34}$$

This conclusion also matches Hoeffding's inequality (Hoeffding, 1994). Yet, we may easily consider other concentration inequalities (Boucheron et al., 2013), or consider other assumptions, to calculate the offset even when the return distribution support is unbounded.

A.3.2 Deriving Guarantees

We note that, as previously claimed, the offset calculated in (A.34) does not depend on the approximated values nor the actual sampled hypotheses, but only on their number m. Thus, using this offset, we can derive "pre-solution" guarantees for the simplified decision problem, as covered in Section 2.2.2. From Lemma 9, we may conclude that using V_m as a simplified value function (i.e., evaluating each candidate with at least m hypotheses) yields a solution which is $(2 \cdot \sqrt{\frac{\ln \frac{1}{\alpha}}{2m}} \cdot (b-a), 2\alpha)$-near optimal, $\forall \alpha \in [0, 0.5]$.

We can also yield (optimized) "post-solution" guarantees, for a specific solution, as previously explained in Section A.2.2. For even greater generality, let us allow every candidate to be evaluated with a different number of hypotheses; let us thus assume π_s^* is evaluated with m_s^* hypotheses, and every other candidate π is evaluated with m_π hypotheses. Now, we recall that when asked to provide guarantees, two types of constraints can be given, and each requires solving a set of minimization problems (A.18)-(A.21), in order to optimize the returned guarantee.

Confidence constraint α assigning the offset from (A.34) in (A.18) means we shall solve the following minimization $\forall \pi \neq \pi_s^*$:

$$\epsilon_\pi^\alpha = \min_{\beta,\gamma} \left\{ \left(\sqrt{\frac{\ln \frac{1}{\beta}}{2m_s^*}} + \sqrt{\frac{\ln \frac{2}{\gamma}}{2m_\pi}} \right) \right\} \cdot (b-a) - \left(V_{m_s^*}(\pi_s^*) - V_{m_\pi}(\pi) \right) \tag{A.35}$$

where

$$0 \leq \beta \leq \alpha \tag{A.36}$$

$$\gamma = \alpha - \beta \tag{A.37}$$

Exploiting the equality constraint leaves us a single degree of freedom (DOF), i.e., an optimization of a single variable. We can see the minimum depends on the proportion of $m_1 : m_2$.

Loss constraint ϵ by inverting the expression we derived in (A.34), which gives the offset for a confidence level, we can extract the confidence level from the offset:

$$\exp\left(-2m_s^* \cdot \left(\frac{\underline{\delta}_\beta(\pi_s^*)}{b-a} \right)^2 \right) = \beta, \tag{A.38}$$

$$\exp\left(-2m_\pi \cdot \left(\frac{\overline{\delta}_\gamma(\pi)}{b-a} \right)^2 \right) = \gamma. \tag{A.39}$$

Assigning these expressions in (A.20) means we shall solve the following minimization $\forall \pi \neq \pi_s^*$:

$$\alpha_\pi^\epsilon = \min_{\underline{\delta}_\beta(\pi_s^*), \overline{\delta}_\gamma(\pi)} \left\{ \exp\left(-2m_s^* \cdot \left(\frac{\underline{\delta}_\beta(\pi_s^*)}{b-a} \right)^2 \right) + \exp\left(-2m_\pi \cdot \left(\frac{\overline{\delta}_\gamma(\pi)}{b-a} \right)^2 \right) \right\} \tag{A.40}$$

where

$$0 \leq \underline{\delta}_\beta(\pi_s^*) \leq \epsilon + \left(V_{m_s^*}(\pi_s^*) - V_{m_\pi}(\pi) \right) \tag{A.41}$$

$$\overline{\delta}_\gamma(\pi) = \epsilon + \left(V_{m_s^*}(\pi_s^*) - V_{m_\pi}(\pi) \right) - \underline{\delta}_\beta(\pi_s^*) \tag{A.42}$$

Again, the equality constraint leaves us a single DOF.

Appendix B

Proofs

The proofs of Lemmas 1,2,3,9 are brought in the main text.

B.1 Lemma 4

Proof.
The properties are trivially given from the definition of action consistency.

□

B.2 Lemma 5

Proof.
Assume f is a monotonously increasing function, such that for every two actions $a_i, a_j \in \mathcal{A}$

$$f(V_1(\boldsymbol{\xi}_1, a_i)) = V_2(\boldsymbol{\xi}_2, a_i), \quad f(V_1(\boldsymbol{\xi}_1, a_j)) = V_2(\boldsymbol{\xi}_2, a_j). \tag{B.1}$$

Then,

$$f(V_1(\boldsymbol{\xi}_1, a_i)) < f(V_1(\boldsymbol{\xi}_1, a_j)) \iff V_2(\boldsymbol{\xi}_2, a_i) < V_2(\boldsymbol{\xi}_2, a_j). \tag{B.2}$$

Because f is monotonously increasing, then $f(x) < f(y) \iff x < y$, and

$$V_1(\boldsymbol{\xi}_1, a_i) < V_1(\boldsymbol{\xi}_1, a_j) \iff V_2(\boldsymbol{\xi}_2, a_i) < V_2(\boldsymbol{\xi}_2, a_j). \tag{B.3}$$

Meaning, $(\boldsymbol{\xi}_1, \mathcal{A}, V_1) \simeq (\boldsymbol{\xi}_2, \mathcal{A}, V_2)$.

Now, to prove the opposite direction, assume $(\boldsymbol{\xi}_1, \mathcal{A}, V_1) \simeq (\boldsymbol{\xi}_2, \mathcal{A}, V_2)$; hence,

$$V_1(\boldsymbol{\xi}_1, a_i) < V_1(\boldsymbol{\xi}_1, a_j) \iff V_2(\boldsymbol{\xi}_2, a_i) < V_2(\boldsymbol{\xi}_2, a_j). \tag{B.4}$$

Let us define a new function f on the domain $\{V_1(\boldsymbol{\xi}_1, a) \mid a \in \mathcal{A}\}$, such that $f(V_1(\boldsymbol{\xi}_1, a)) \doteq V_2(\boldsymbol{\xi}_2, a)$. Given this definition, and the action consistency conditions from (B.4), we can conclude that

$$\begin{aligned} f(V_1(\boldsymbol{\xi}_1, a_i)) < f(V_1(\boldsymbol{\xi}_1, a_j)) \\ \iff V_2(\boldsymbol{\xi}_2, a_i) < V_2(\boldsymbol{\xi}_2, a_j) \iff \\ V_1(\boldsymbol{\xi}_1, a_i) < V_1(\boldsymbol{\xi}_1, a_j). \end{aligned} \tag{B.5}$$

Thus, f is monotonously increasing on its domain.

□

B.3 Lemma 6

Proof.
Both directions are a direct consequence of Lemma 5. Assume $\Delta^*(\mathcal{P}, \mathcal{P}_s) = 0$. Thus, a monotonously increasing balance function f exists, such that $\Delta(\mathcal{P}, \mathcal{P}_s^f) = 0$. Meaning, for every action $a \in \mathcal{A}$, $f(V_s(\boldsymbol{\xi}_s, a)) = V(\boldsymbol{\xi}, a)$. According to Lemma 5, it is sufficient to prove that $\mathcal{P} \simeq \mathcal{P}_s$.

To prove the opposite direction, assume $\mathcal{P} \simeq \mathcal{P}_s$. Let us define a new function f on the domain $\{V_s(\boldsymbol{\xi}_s, a) \mid a \in \mathcal{A}\}$, such that $f(V_s(\boldsymbol{\xi}_s, a)) \doteq V(\boldsymbol{\xi}, a)$. From this definition, $\Delta(\mathcal{P}, \mathcal{P}_s^f) = 0$. Also, according to Lemma 5, this function f is monotonously increasing, and can be used to conclude that $\Delta^*(\mathcal{P}, \mathcal{P}_s) = 0$.

□

B.4 Lemma 7

Proof.
From the definition of the simplification offset, we know that for every monotonously increasing function f, the following is true:

$$|V(\boldsymbol{\xi}, a^*) - f(V_s(\boldsymbol{\xi}_s, a^*))| \leq \Delta(\mathcal{P}, \mathcal{P}_s^f), \tag{B.6}$$
$$|V(\boldsymbol{\xi}, a_s^*) - f(V_s(\boldsymbol{\xi}_s, a_s^*))| \leq \Delta(\mathcal{P}, \mathcal{P}_s^f). \tag{B.7}$$

Removing the absolute values surely does not compromise the inequalities:

$$V(\boldsymbol{\xi}, a^*) - f(V_s(\boldsymbol{\xi}_s, a^*)) \leq \Delta(\mathcal{P}, \mathcal{P}_s^f), \tag{B.8}$$
$$f(V_s(\boldsymbol{\xi}_s, a_s^*)) - V(\boldsymbol{\xi}, a_s^*) \leq \Delta(\mathcal{P}, \mathcal{P}_s^f). \tag{B.9}$$

By adding the two inequalities, and utilizing the definition of the *loss*, we get:

$$loss(\mathcal{P}, \mathcal{P}_s^f) + f(V_s(\boldsymbol{\xi}_s, a_s^*)) - f(V_s(\boldsymbol{\xi}_s, a^*)) \leq 2 \cdot \Delta(\mathcal{P}, \mathcal{P}_s^f). \tag{B.10}$$

From the definition of a_s^*, we know that

$$V_s(\boldsymbol{\xi}_s, a_s^*)) \geq V_s(\boldsymbol{\xi}_s, a^*). \tag{B.11}$$

Since f is monotonously increasing, then also

$$f(V_s(\boldsymbol{\xi}_s, a_s^*))) \geq f(V_s(\boldsymbol{\xi}_s, a^*)), \tag{B.12}$$
$$f(V_s(\boldsymbol{\xi}_s, a_s^*))) - f(V_s(\boldsymbol{\xi}_s, a^*)) \geq 0. \tag{B.13}$$

Thus, we can infer that

$$loss(\mathcal{P}, \mathcal{P}_s^f) \leq 2 \cdot \Delta(\mathcal{P}, \mathcal{P}_s^f). \tag{B.14}$$

Since the final statement is true for any monotonously increasing function f, we may conclude the desired upper bound over the loss,

$$loss(\mathcal{P}, \mathcal{P}_s) \leq 2 \cdot \Delta^*(\mathcal{P}, \mathcal{P}_s) \tag{B.15}$$

□

B.5 Lemma 8

Proof.
Let us examine three decision problems $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, where $\mathcal{P}_i \doteq (\boldsymbol{\xi}_i, \mathcal{A}, V_i)$. Now, for each two problems $\mathcal{P}_i, \mathcal{P}_j$, we mark $a_{ij} \in \mathcal{A}$ as the action, and f_{ij} as the function, for which $\Delta^*(\mathcal{P}_i, \mathcal{P}_j) \doteq \delta(\mathcal{P}_i, \mathcal{P}_j^{f_{ij}}, a_{ij})$ (the values can be chosen arbitrarily from all values which comply to the equation). According to this notation we can conclude:

$$\begin{aligned}
&\Delta^*(\mathcal{P}_1, \mathcal{P}_2) + \Delta^*(\mathcal{P}_2, \mathcal{P}_3) \doteq \\
&\delta(\mathcal{P}_1, \mathcal{P}_2^{f_{12}}, a_{12}) + \delta(\mathcal{P}_2, \mathcal{P}_3^{f_{23}}, a_{23}) \geq \\
&\delta(\mathcal{P}_1, \mathcal{P}_2^{f_{12}}, a_{13}) + \delta(\mathcal{P}_2, \mathcal{P}_3^{f_{23}}, a_{13}) \doteq \\
&|V_1(\boldsymbol{\xi}_1, a_{13}) - f_{12}(V_2(\boldsymbol{\xi}_2, a_{13}))| + |V_2(\boldsymbol{\xi}_2, a_{13}) - f_{23}(V_3(\boldsymbol{\xi}_3, a_{13}))| \geq \\
&|V_1(\boldsymbol{\xi}_1, a_{13}) - f_{12}(V_2(\boldsymbol{\xi}_2, a_{13})) + V_2(\boldsymbol{\xi}_2, a_{13}) - f_{23}(V_3(\boldsymbol{\xi}_3, a_{13}))| \doteq (\star\star).
\end{aligned} \tag{B.16}$$

Let us define the following scalar function:

$$F(x) \doteq f_{23}(x) + f_{12}(V_2(\boldsymbol{\xi}_2, a_{13})) - V_2(\boldsymbol{\xi}_2, a_{13}) = f_{23}(x) + \text{constant}. \tag{B.17}$$

Since f_{23} is a monotonously increasing, so is F, and

$$\begin{aligned}
(\star\star) = |V_1(\boldsymbol{\xi}_1, a_{13}) - F(V_3(\boldsymbol{\xi}_3, a_{13}))| \doteq \\
\Delta(\mathcal{P}_1, \mathcal{P}_3^F, a_{13}) \geq \\
\Delta(\mathcal{P}_1, \mathcal{P}_3^{f_{13}}, a_{13}) = \\
\Delta^*(\mathcal{P}_1, \mathcal{P}_3).
\end{aligned} \tag{B.18}$$

Hence, Δ^* satisfies the triangle inequality.

□

B.6 Corollary 1

Proof.
Let us mark as $\boldsymbol{R}_s^p$ the sparsified square root matrix, before permuting the variables back to their original order in line 8 of Algorithm 3. First, we show that applying the reverse permutation $\boldsymbol{P}\square\boldsymbol{P}^T$ on $\boldsymbol{R}_s^p$ indeed leads to a square root of the sparse information matrix $\boldsymbol{\Lambda}_s$ (in the original order):

$$(\boldsymbol{P}\boldsymbol{R}_s^p\boldsymbol{P}^T)^T(\boldsymbol{P}\boldsymbol{R}_s^p\boldsymbol{P}^T) = \boldsymbol{P}\boldsymbol{R}_s^{pT}\boldsymbol{R}_s^p\boldsymbol{P}^T = \boldsymbol{P}\boldsymbol{\Lambda}_s^p\boldsymbol{P}^T = \boldsymbol{\Lambda}_s, \tag{B.19}$$

where $\boldsymbol{\Lambda}_s^p$ is the sparsified information matrix, before permuting the variables back.

Now, we want to examine the shape of the matrix $\boldsymbol{R}_s \doteq \boldsymbol{P}\boldsymbol{R}_s^p\boldsymbol{P}^T$, and show that it is indeed triangular. According to the Algorithm 3, before executing line 8, $\boldsymbol{R}_s^p$ is of the following structure:

$$\boldsymbol{R}_s^p = \left(\begin{array}{c|c} \text{diagonal} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \text{triangular} \end{array}\right), \tag{B.20}$$

where the rows of the diagonal block correspond to the sparsified variables. Without losing generality, we should only prove that applying a permutation of the form p': $(1, \ldots, n) \mapsto (2, \ldots, i, 1, i+1, \ldots, n)$ on this matrix (i.e., "pushing forwards" one of the sparsified variables), does not break the triangular form. Hence, assuming $\boldsymbol{P}^T$ is the column permutation matrix matching such p', let us look at

$$
\begin{aligned}
\boldsymbol{R}_s \doteq \boldsymbol{P}\boldsymbol{R}_s^p\boldsymbol{P}^T = \\
&\boldsymbol{P}\left(\begin{array}{c|c}
d \in \mathbb{R} & 0\dots0 \\ \hline
0 & \\
\vdots & \text{triangular} \\
0 &
\end{array}\right)\boldsymbol{P}^T = \\
&\left(\begin{array}{c|c|c}
0 & & \\
\vdots & \text{triangular} & * \\
0 & & \\ \hline
d & 0\dots0 & 0\dots0 \\ \hline
0 & & \\
\vdots & \mathbf{0} & \text{triangular} \\
0 & &
\end{array}\right)\boldsymbol{P}^T = \\
&\left(\begin{array}{c|c|c}
 & 0 & \\
\text{triangular} & \vdots & * \\
 & 0 & \\ \hline
0\dots0 & d & 0\dots0 \\ \hline
 & 0 & \\
\mathbf{0} & \vdots & \text{triangular} \\
 & 0 &
\end{array}\right).
\end{aligned}
\tag{B.21}
$$

Recursively utilizing this conclusion, for more intricate permutations, proves that $\boldsymbol{R}_s$ is indeed triangular, whenever permuting the sparsified variables back to their original order, as desired. □

B.7 Theorem 1

Proof.
Consider a belief $b = \mathcal{N}(\boldsymbol{X}*, \boldsymbol{\Lambda}^{-1})$, where the state contains n_1 uninvolved variables and n_2 involved variables, such that $n = n_1 + n_2$ is the prior state size. Also consider the simplified belief $b_s = \mathcal{N}(\boldsymbol{X}^*, \boldsymbol{\Lambda}_s^{-1})$, in which all uninvolved variables were sparsified, by applying Algorithm 3.

We mark with $\boldsymbol{P}$ the (column) permutation matrix that positions all the involved variable at the end of the state. Now, let $\boldsymbol{R}^p$ be the Cholesky factor of the permuted information matrix $\boldsymbol{\Lambda}^p \doteq \boldsymbol{P}^T\boldsymbol{\Lambda}\boldsymbol{P}$, such that $\boldsymbol{\Lambda}^p = \boldsymbol{R}^{pT}\boldsymbol{R}^p$. This $\boldsymbol{R}^p$ can be divided into block form:

$$
\boldsymbol{R}^p \doteq \left(\begin{array}{c|c}
\boldsymbol{R}_{11}^p & \boldsymbol{R}_{12}^p \\ \hline
\mathbf{0}^{n_2\times n_1} & \boldsymbol{R}_{22}^p
\end{array}\right),
\tag{B.22}
$$

where $\boldsymbol{R}_{11}^p \in \mathbb{R}^{n_1\times n_1}$ and $\boldsymbol{R}_{22}^p \in \mathbb{R}^{n_2\times n_2}$ are triangular sub-matrices, $\boldsymbol{R}_{12}^p \in \mathbb{R}^{n_1\times n_2}$, and $\mathbf{0}^{n_1\times n_2}$ is a zero matrix in the specified size. By following the steps of Algorithm 3, we realize that the returned sparsified information matrix $\boldsymbol{\Lambda}_s$ is given as $\boldsymbol{\Lambda}_s \doteq \boldsymbol{P}\boldsymbol{R}_s^{pT}\boldsymbol{R}_s^p\boldsymbol{P}^T$ (or, equally, satisfies $\boldsymbol{P}^T\boldsymbol{\Lambda}_s\boldsymbol{P} \doteq \boldsymbol{R}_s^{pT}\boldsymbol{R}_s^p$), where

$$
\boldsymbol{R}_s^p \doteq \left(\begin{array}{c|c}
\boldsymbol{D}_{11}^p & \mathbf{0}^{n_1\times n_2} \\ \hline
\mathbf{0}^{n_2\times n_1} & \boldsymbol{R}_{22}^p
\end{array}\right),
\tag{B.23}
$$

and $\boldsymbol{D}_{11}^{p}$ is the diagonal matrix formed by copying the diagonal of $\boldsymbol{R}_{11}^{p}$ (and assigning zero elsewhere).

We would like to find the simplification offset between the two decision problems $\mathcal{P}$ and $\mathcal{P}_s$ (for which b and b_s are the initial beliefs, respectively). Let us consider a candidate policy $\pi \in \Pi$. Then, we may derive the following from the definition of the offset and the objective function $\tilde{V}$:

$$\delta(\mathcal{P}, \mathcal{P}_s, \pi) \doteq \left|\tilde{V}(b,\pi) - \tilde{V}(b_s,\pi)\right| = \tag{B.24}$$

$$\left|\mathop{\mathbb{E}}_{H}[\mathrm{H}(b) - \mathrm{H}(b_H)] - \mathop{\mathbb{E}}_{H}[\mathrm{H}(b_s) - \mathrm{H}(b_{sH})]\right| = \tag{B.25}$$

$$\left|\mathop{\mathbb{E}}_{H}[\mathrm{H}(b_H) - \mathrm{H}(b_{sH})]\right| = \tag{B.26}$$

$$\frac{1}{2}\cdot\left|\mathop{\mathbb{E}}_{H}[\ln|\boldsymbol{\Lambda}_H| - \ln|\boldsymbol{\Lambda}_{sH}|]\right|, \tag{B.27}$$

where $\boldsymbol{\Lambda}_H$ and $\boldsymbol{\Lambda}_{sH}$ represent the posterior information matrices, after updating of the respective priors, according to the hypothesis H (for the candidate π). For ease of writing, let us use $\boldsymbol{U} \in \mathbb{R}^{h\times(n+m)}$ to mark the collective Jacobian corresponding to H (where $n+m$ is the posterior state size); hence, from the additivity of the information (3.24), the last expression can be written as

$$\delta(\mathcal{P}, \mathcal{P}_s, \pi) = \frac{1}{2}\cdot\left|\mathbb{E}\left[\ln\left|\breve{\boldsymbol{\Lambda}} + \boldsymbol{U}^T\boldsymbol{U}\right| - \ln\left|\breve{\boldsymbol{\Lambda}}_s + \boldsymbol{U}^T\boldsymbol{U}\right|\right]\right|, \tag{B.28}$$

where

$$\breve{\boldsymbol{\Lambda}} \doteq \left(\begin{array}{c|c} \boldsymbol{\Lambda} & \boldsymbol{0}^{n\times m} \\ \hline \boldsymbol{0}^{m\times n} & \boldsymbol{0}^{m\times m} \end{array}\right) \tag{B.29}$$

is the augmented information matrix (and similarly for $\boldsymbol{\Lambda}_s$).

Now, let us examine the following expression:

$$\# \doteq \left|\breve{\boldsymbol{\Lambda}} + \boldsymbol{U}^T\boldsymbol{U}\right| - \left|\breve{\boldsymbol{\Lambda}}_s + \boldsymbol{U}^T\boldsymbol{U}\right|, \tag{B.30}$$

We know that (unitary) variable permutation does not affect the determinant of a matrix, thus

$$\begin{aligned} \# = \left|\breve{\boldsymbol{P}}^T\left(\breve{\boldsymbol{\Lambda}} + \boldsymbol{U}^T\boldsymbol{U}\right)\breve{\boldsymbol{P}}\right| - \left|\breve{\boldsymbol{P}}^T\left(\breve{\boldsymbol{\Lambda}}_s + \boldsymbol{U}^T\boldsymbol{U}\right)\breve{\boldsymbol{P}}\right| = \\ \left|\breve{\boldsymbol{P}}^T\breve{\boldsymbol{\Lambda}}\breve{\boldsymbol{P}} + (\boldsymbol{U}\breve{\boldsymbol{P}})^T(\boldsymbol{U}\breve{\boldsymbol{P}})\right| - \left|\breve{\boldsymbol{P}}^T\breve{\boldsymbol{\Lambda}}_s\breve{\boldsymbol{P}} + (\boldsymbol{U}\breve{\boldsymbol{P}})^T(\boldsymbol{U}\breve{\boldsymbol{P}})\right|, \end{aligned} \tag{B.31}$$

where

$$\breve{\boldsymbol{P}} \doteq \left(\begin{array}{c|c} \boldsymbol{P} & \boldsymbol{0}^{n\times m} \\ \hline \boldsymbol{0}^{m\times n} & \boldsymbol{I}^{m\times m} \end{array}\right) \tag{B.32}$$

is the augmented permutation matrix, which keeps the variables added in the update at the end of the state. Note that if the variables were not originally added to the end of the state, the permutation $\breve{\boldsymbol{P}}$ can be easily adapted to enforce this property.

We can also augment the matrix $\boldsymbol{R}^p$ with m empty columns (and similarly for $\boldsymbol{R}_s^p$):

$$\breve{\boldsymbol{R}}^p \doteq \left(\begin{array}{c|c|c} \boldsymbol{R}_{11}^p & \boldsymbol{R}_{12}^p & \boldsymbol{0}^{n\times m} \\ \hline \boldsymbol{0}^{n_2\times n_1} & \boldsymbol{R}_{22}^p & \end{array}\right), \tag{B.33}$$

and assign the result in #, to yield:

$$\# = \left|\breve{\boldsymbol{R}}^{p\,T}\breve{\boldsymbol{R}}^p + (\boldsymbol{U}\breve{\boldsymbol{P}})^T(\boldsymbol{U}\breve{\boldsymbol{P}})\right| - \left|\breve{\boldsymbol{R}}_s^{p\,T}\breve{\boldsymbol{R}}_s^p + (\boldsymbol{U}\breve{\boldsymbol{P}})^T(\boldsymbol{U}\breve{\boldsymbol{P}})\right| \tag{B.34}$$

This expression can be reorganized to the following form:

$$\# = \left| \begin{pmatrix} \breve{\boldsymbol{R}}^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix}^T \begin{pmatrix} \breve{\boldsymbol{R}}^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix} \right| - \left| \begin{pmatrix} \breve{\boldsymbol{R}}_s^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix}^T \begin{pmatrix} \breve{\boldsymbol{R}}_s^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix} \right| \tag{B.35}$$

The two matrices which appear in this expression also follow a block form:

$$\begin{pmatrix} \breve{\boldsymbol{R}}^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix} = \left(\begin{array}{c|c|c} \boldsymbol{R}_{11}^p & \boldsymbol{R}_{12}^p & \boldsymbol{0}^{n_1 \times m} \\ \hline \boldsymbol{0}^{(n_2+h)\times n_1} & \multicolumn{2}{c}{\boldsymbol{B}} \end{array} \right), \tag{B.36}$$

$$\begin{pmatrix} \breve{\boldsymbol{R}}_s^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix} = \left(\begin{array}{c|c} \boldsymbol{D}_{11}^p & \boldsymbol{0}^{n_1 \times (n_2+m)} \\ \hline \boldsymbol{0}^{(n_2+h)\times n_1} & \boldsymbol{B} \end{array} \right), \tag{B.37}$$

where

$$\boldsymbol{B} \doteq \left(\begin{array}{c|c} \boldsymbol{R}_{22}^p & \boldsymbol{0}^{n_2 \times m} \\ \hline \multicolumn{2}{c}{\boldsymbol{U}^{\text{inv}}} \end{array} \right), \tag{B.38}$$

and $\boldsymbol{U}^{\text{inv}}$ is a sub-matrix of $\boldsymbol{U}\breve{\boldsymbol{P}}$, containing its right $n_2 + m$ columns. Since the left n_1 columns of $\boldsymbol{U}\breve{\boldsymbol{P}}$ correspond to uninvolved variables, we know they may only contain zeros.

Thus, if we mark $\breve{\boldsymbol{R}}_{12}^p \doteq \left(\begin{array}{c|c} \boldsymbol{R}_{12}^p & \boldsymbol{0}^{n_1 \times m} \end{array} \right)$, then the left term in (B.35) is:

$$\left| \begin{pmatrix} \breve{\boldsymbol{R}}^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix}^T \begin{pmatrix} \breve{\boldsymbol{R}}^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix} \right| = \left| \begin{array}{c|c} {\boldsymbol{R}_{11}^p}^T \boldsymbol{R}_{11}^p & {\boldsymbol{R}_{11}^p}^T \breve{\boldsymbol{R}}_{12}^p \\ \hline {\breve{\boldsymbol{R}}_{12}^p}^T \boldsymbol{R}_{11}^p & {\breve{\boldsymbol{R}}_{12}^p}^T \breve{\boldsymbol{R}}_{12}^p + \boldsymbol{B}^T\boldsymbol{B} \end{array} \right|. \tag{B.39}$$

From the block-determinant formula (see Harville, 1998), this equals to

$$\left| {\boldsymbol{R}_{11}^p}^T \boldsymbol{R}_{11}^p \right| \cdot \left| {\breve{\boldsymbol{R}}_{12}^p}^T \breve{\boldsymbol{R}}_{12}^p + \boldsymbol{B}^T\boldsymbol{B} - {\breve{\boldsymbol{R}}_{12}^p}^T \boldsymbol{R}_{11}^p {\boldsymbol{R}_{11}^p}^{-1} {\boldsymbol{R}_{11}^p}^{T^{-1}} {\boldsymbol{R}_{11}^p}^T \breve{\boldsymbol{R}}_{12}^p \right| = \left|\boldsymbol{R}_{11}^p\right|^2 \cdot \left|\boldsymbol{B}^T\boldsymbol{B}\right| \tag{B.40}$$

The right term in (B.35) is:

$$\left| \begin{pmatrix} \breve{\boldsymbol{R}}_s^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix}^T \begin{pmatrix} \breve{\boldsymbol{R}}_s^p \\ \boldsymbol{U}\breve{\boldsymbol{P}} \end{pmatrix} \right| = \left| \begin{array}{c|c} {\boldsymbol{D}_{11}^p}^T \boldsymbol{D}_{11}^p & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{B}^T\boldsymbol{B} \end{array} \right| = \left| {\boldsymbol{D}_{11}^p}^2 \right| \cdot \left|\boldsymbol{B}^T\boldsymbol{B}\right| \tag{B.41}$$

Since $\boldsymbol{R}_{11}^p$ and $\boldsymbol{D}_{11}^p$ are triangular matrices with the same diagonal, their determinants are equal (to the product of the diagonal elements). Thus, $\# = 0$, and overall

$$\left|\breve{\boldsymbol{\Lambda}} + \boldsymbol{U}^T\boldsymbol{U}\right| = \left|\breve{\boldsymbol{\Lambda}}_s + \boldsymbol{U}^T\boldsymbol{U}\right|. \tag{B.42}$$

This surely means that

$$\ln\left|\breve{\boldsymbol{\Lambda}} + \boldsymbol{U}^T\boldsymbol{U}\right| - \ln\left|\breve{\boldsymbol{\Lambda}}_s + \boldsymbol{U}^T\boldsymbol{U}\right| = 0. \tag{B.43}$$

Finally, assigning this expression in (B.28) means that

$$\delta(\mathcal{P}, \mathcal{P}_s, \pi) = 0. \tag{B.44}$$

Since the previous conclusion is true $\forall \pi \in \Pi$, this means that

$$\Delta(\mathcal{P}, \mathcal{P}_s) \doteq \max_{\pi \in \Pi} \delta(\mathcal{P}, \mathcal{P}_s, \pi) = 0, \tag{B.45}$$

as desired.

□

I

תקציר

היעד העיקרי של חקר בינה מלאכותית ורובוטיקה הוא לאפשר לסוכנים ורובוטים לתכנן ולבצע פעולות באופן עצמאי. על מנת להשיג ביצועים מהימנים, על סוכנים אלה להתחשב באי-ודאות המצויה בעולם האמיתי; אי-ודאות זו יכולה לנבוע, למשל, מפעילות בסביבות דינמיות, מחיישנים רועשים, או מביצוע לא מדויק של פעולות. באופן מעשי, נתונים אלה דורשים לבחון מצבים הסתברותיים, הידועים כ"אמונות". אמונות שכאלה הן, למעשה, התפלגויות מעל לווקטורי מצב, המבטאות את הידע של הסוכן על עצמו וסביבתו. באופן כללי, מטרתה של עבודתנו זו היא לאפשר קבלת החלטות יעילה תחת אי-ודאות, אותה אנחנו מנסחים כקבלת החלטות ב"מרחב האמונה". כדי להגדיר את אופן הפעולה המיטבי, על הסוכן לצפות את ההתפתחות העתידית של אמונתו ההסתברותית, ולעדכן אותה בהתאם לכך, בהתחשב בפעולות מועמדות מרובות. בחינת אמונות מעל וקטורי מצב ממימד גבוה (המבטאים, למשל, התפלגויות מעל מסלולים מלאים) יכולה לשפר את דיוק השיערוך באופן משמעותי. אמנם, שיפור זה מגיע, כמובן, על חשבון מורכבות חישובית גבוהה, הגורמת לכך שפתרון בעיות החלטה שכאלה בזמן-אמת הינו אתגר משמעותי. מטבע הדברים, פתרון בעיית ההחלטה מצריך לאמוד את הפעולות המועמדות תחת פונקציית מטרה כלשהי. בעבודה זו אנו טוענים שלעיתים ניתן לותר על חישוב מדויק של ערכי הפעולות, ובמקום זאת, לזהות ולפתור בעיית החלטה מפושטת, אשר מובילה לבחירת פעולה זהה (או דומה). ניתן לפשט בעיית החלטה על ידי שינוי של כל אחד ממרכיביה – האמונה ההתחלתית, פונקציית המטרה או קבוצת הפעולות המועמדות – כך שחישוב פונקציית המטרה יעשה קל יותר. כדי לתמוך ברעיון זה, אנו מציגים מספר תרומות. אנו מתחילים בהצגת תשתית תאורטית חדישה ויסודית, אשר מאפשרת לנתח באופן רשמי את המיטביות של פעולות, ולכמת את ההפסד שעלול להיגרם מבחירה של פעולה שאינה מיטבית. שיטת הניתוח הזו גם מאפשרת לנו לזהות פעולות שאינן מיטביות באופן מובטח, ועל כן ניתנות לסילוק. לאחר מכן אנו מראים כיצד ניתן ליישם את הרעיון של פישוט בעיות בהקשר של קבלת החלטות במרחב האמונה. באופן מעשי, האמונה לרוב מיוצגת באמצעות מודל גרפי, כגון גרף אילוצים, עבור אמונות מהתפלגות כללית, או באמצעות "מטריצת השורש הריבועי" שהינה משולשת עליונה וממימד גבוה, עבור התפלגויות נורמליות. לפיכך, אנו מציעים לפשט את הבעיה על ידי בחינה של קירוב דליל של האמונה ההתחלתית (קרי, דילול של הגרף או המטריצה המייצגת). קירוב שכזה ניתן לעדכון באופן יעיל יותר, בעת בחינת הפעולות המועמדות. בהתאם לכך, אנו מציגים אלגוריתם מדרגי לדילול אמונות, ומנתחים את השפעתו על בעיית ההחלטה באמצעות התשתית המוצעת. מניתוח זה ניתן להבטיח שבמקרים מסוימים דילול שכזה כלל אינו גורם להקרבה באיכות הפתרון. אנו ממשיכים בהצגה של שיטה נוספת להפחתת המורכבות החישובית של הבעיה, על ידי אופטימיזציה של סדר המשתנים בווקטור המצב, לפני קבלת ההחלטות. פעולה זו יכולה לעזור בהפחתת מספר המשתנים שיושפעו בעת עדכון האמונה. אנו מתייחסים לשיטה זו כ"טקטיקת סידור משתנים הדרגתית נבואית", שכן סידור המשתנים נקבע מראש, בהתאם להתפתחות האמונה אותה אנו צופים בעתיד. למרבה הצער, שינוי סדר המשתנים גורם לשינוי מטריצת השורש הריבועי באופן שאינו מובן מאליו, ומרמז על הצורך בחישובה מחדש. מכאן, על מנת לתמוך בשיטה זו, אנו מציגים אלגוריתם נוסף, יעיל ומקבילי, לתיקון ישיר של המטריצה בעת שינוי סדר המשתנים. אנו מדגימים את היתרונות של השיטות המוצעות בפתרון בעיות ניווט ומיפוי על ידי רובוטים ניידים, בהן אנו מצליחים להביא להפחתה משמעותית בזמן החישוב, ללא פגיעה מעשית באיכות הפתרון. בעיות רלוונטיות נוספות כוללות מיקום חיישנים, נהיגה אוטונומית ומניפולציה באמצעות רובוטים.

המחקר נעשה בהנחיית פרופסור חבר ואדים אינדלמן
במסגרת התוכנית הבין-יחידתית למערכות אוטונומיות ורובוטיקה

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי

קבלת החלטות יעילה תחת אי-ודאות במרחבי מצב ממימד גבוה

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר דוקטור לפילוסופיה

חן אלימלך

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
תמוז תשפ"א, חיפה, יוני 2021