Introducing PIVOT: Predictive Incremental Variable Ordering Tactic for Efficient Belief Space Planning

Khen Elimelech[1] and Vadim Indelman[2]

[1] Robotics and Autonomous Systems Program,
[2] Department of Aerospace Engineering,
Technion - Israel Institute of Technology, Haifa 32000, Israel.
{khen,vadim.indelman}@technion.ac.il

Abstract. Belief Space Planning (BSP) is a fundamental technique in artificial intelligence and robotics, which is widely used in the solution of problems such as online autonomous navigation and manipulation. Unfortunately, BSP is computationally demanding, especially when dealing with high-dimensional state spaces. We thus introduce PIVOT: Predictive Incremental Variable Ordering Tactic, a novel approach to improve planning efficiency. Although variable ordering has been extensively used for the state inference problem, variable ordering specifically for planning has hardly been considered. Interestingly, this tactic can also lead to improved loop-closing efficiency during state inference. We use the approach in an active-SLAM scenario, and demonstrate a significant improvement in efficiency. This approach follows our previous work regarding efficient BSP via belief sparsification.

1 Introduction

1.1 Problem Overview

The task of online decision making under uncertainty is often modeled and solved as a Belief Space Planning (BSP) problem [2]. To solve this problem, we shall first maintain the posterior distribution (the "belief") over an agent's state, considering the sequence of noisy past actions and sensor observations, in a process known as state inference. State-of-the-art state inference approaches (e.g. [14,13,11]) rely on incrementally updating the upper-triangular square root information matrix of the belief, with the arrival of new constraints. Such incremental updates are efficient when new constraints involve only recent state variables, but can still grow very expensive when involving variables from the past (loop closures). In planning, we shall examine the predicted propagation of this belief considering several candidate actions, under some objective measure; as this essentially means solving multiple multi-step state inference sessions, possibly containing loop closures, BSP can turn computationally challenging for online solution. BSP is used to solve prominent tasks such as active simultaneous localization and mapping (SLAM), and robotic manipulation, and thus reducing its cost is of great importance.

1.2 Contribution and Related Work

Various methods were introduced to deal with the high computational cost of the planning problem. These include, e.g., sampling methods [19]; local trajectory optimization [16]; and point-value iteration [18]. In this work, we investigate a more fundamental approach, intended to improve the process of belief update, which is required when evaluating candidate actions (however they are defined). Inherently, such approach is complementary to other planning methods, such as those mentioned before. We thus present the Predictive Incremental Variable Ordering Tactic (PIVOT). With this method, we suggest to perform a precursory variable reordering procedure on the belief, in order to optimize the number of variables to update in planning. When considering sequential re-planning, the order can be applied incrementally before each planning session. The approach is highly relevant to high-dimensional state spaces; we demonstrate so in this paper by applying it in an active-SLAM scenario, showing a substantial improvement in efficiency.

This idea is a sequel to our previous work regarding sparsification for efficent decision making [7,8,9]. Formerly, a "batch" sparsification procedure was conducted on the belief before planning, without affecting the maintained state in inference. The newly suggested procedure, however, is performed on the inferred state, and not "forgotten" after every planning session, which allows us to update it incrementally. Interestingly, this also allows our method to improve loop-closing efficiency during state inference. Still, the approach is only relevant in the context of planning. The predictive order relies on the candidate actions as defined in planning. Also, precursory reordering only makes sense when applying multiple actions on the initial belief; otherwise, the reordering can be combined with the update itself.

Although variable ordering has been extensively used for the state inference problem, variable ordering specifically for planning, as presented here, has hardly been considered. As explained, to solve the problem, we examine the propagated belief's triangular square root information matrix. It is widely known that the variable order in the state can have a dramatic impact on the number of non-zeros in the root matrix. Thus, fill-reducing variable ordering tactics are very commonly used in the solution of the state inference problem, especially for SLAM [1]. The aforementioned inference techniques, iSAM2 [13] and SLAM++ [11], apply such orders incrementally, during belief update. Unlike these standard ordering tactics, which arrange the variables according to the current information, we suggest to order the variables based on predicted future development. One work which indeed considers variable ordering in the context of planning was introduced in [3]. This approach utilizes variable ordering to allow reuse of calculations for similar actions and successive planning iterations. This scheme is conceptually different from the one presented here, and also requires external caching. Also, that work is problem-specific, while we impose no such restriction.

2 Preliminaries

2.1 Belief Space Planning

We consider a sequential POMDP process. At time-step k, an agent at pose $\boldsymbol{x}_{k-1}$ transitions to pose $\boldsymbol{x}_k$, using a control $\boldsymbol{u}_k$, and then collects an observation of the world $\boldsymbol{z}_k$. The agent's trajectory $\boldsymbol{x}_{0:k}$, alongside an optional vector of external variables $\boldsymbol{l}$, are collected in the state vector

$$\boldsymbol{X}_k = \begin{bmatrix} \boldsymbol{x}_{0:k} \\ \boldsymbol{l} \end{bmatrix}, \quad \text{where} \quad \boldsymbol{x}_{0:k} \doteq \begin{bmatrix} \boldsymbol{x}_0 \\ \vdots \\ \boldsymbol{x}_k \end{bmatrix}. \tag{1}$$

The transition and observation models are both contain some Gaussian noise, and are described as

$$\boldsymbol{x}_k = g_k(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k, \quad \boldsymbol{w}_k \sim \mathcal{N}(0, \boldsymbol{W}_k), \tag{2}$$

$$\boldsymbol{z}_k = h_k(\boldsymbol{X}_k) + \boldsymbol{v}_k, \quad \boldsymbol{v}_k \sim \mathcal{N}(0, \boldsymbol{V}_k), \tag{3}$$

where $\boldsymbol{W}_k, \boldsymbol{V}_k$ are the covariance matrices of the respective zero-mean Gaussian noise $\boldsymbol{w}_k, \boldsymbol{v}_k$, and g_k and h_k are deterministic functions.

At each time-step, the agent maintains a posterior distribution over its state vector, given the controls and observations taken until that time. This distribution is the agent's *belief*. By linearizing the models, given an initial state estimate $\overline{\boldsymbol{X}}_k$, we can estimate the belief at time-step k

$$b_k(\boldsymbol{X}_k) \doteq \mathbb{P}(\boldsymbol{X}_k \mid u_{1:k}, z_{1:k}) \approx \mathcal{N}(\boldsymbol{X}_k^*, \boldsymbol{\Lambda}_k^{-1}), \tag{4}$$

where $\boldsymbol{G}_k \doteq \nabla|_{\overline{\boldsymbol{X}}_k} g_k$, $\boldsymbol{H}_k \doteq \nabla|_{\overline{\boldsymbol{X}}_k} h_k$, and $\boldsymbol{\Lambda}_k$ is the information matrix, achieved by calculating

$$\boldsymbol{\Lambda}_k \doteq \boldsymbol{U}_{0:k}^T \boldsymbol{U}_{0:k}, \quad \text{where} \quad \boldsymbol{U}_{0:k} \doteq \begin{bmatrix} \boldsymbol{U}_0 \\ \vdots \\ \boldsymbol{U}_k \end{bmatrix}, \quad \boldsymbol{U}_{k\geq 1} \doteq \begin{bmatrix} \boldsymbol{W}_k^{-\frac{1}{2}} \cdot \boldsymbol{G}_k \\ \boldsymbol{V}_k^{-\frac{1}{2}} \cdot \boldsymbol{H}_k \end{bmatrix}, \tag{5}$$

$\boldsymbol{U}_0$ is the prior information root, and $\boldsymbol{X}_k^*$ is the maximum a posteriori (MAP) state estimate. In this least-squares approach, this MAP estimate is gained by (iteratively) calculating its delta from the previous estimate. This "back-substitution" calculation requires the upper triangular square root of the information matrix $\boldsymbol{R}_k$, such that $\boldsymbol{\Lambda}_k \doteq \boldsymbol{R}_k^T \boldsymbol{R}_k$, given by the Cholesky or QR factorization (for more details see [6]). When applying a sequence of T additional controls $u \doteq u_{k+1:k+T}$, the information matrix is simply updated according to:

$$\boldsymbol{\Lambda}_{k+T} = \boldsymbol{\Lambda}_k + \boldsymbol{U}^T \boldsymbol{U}, \quad \text{where} \quad \boldsymbol{U} \doteq \boldsymbol{U}_{k+1:k+T} = \begin{bmatrix} \boldsymbol{U}_{k+1} \\ \vdots \\ \boldsymbol{U}_{k+T} \end{bmatrix}. \tag{6}$$

We call U the *collective Jacobian* of the control sequence u. Conviniently, $\boldsymbol{R}_{k+T}$ can be incrementally calculated from $\boldsymbol{R}_k$, instead of calculating the factorization of the new information matrix from scratch. For example, according to [11], it is enough to recalculate the Cholesky factorization of only the affected bottom-right block of $\boldsymbol{R}_{k+T}$, starting from the index of the first non-zero column of $\boldsymbol{U}$; the variables in this block are the *affected variables.* Alternatively, [14] also identifies the affected variables, but instead utilizes the QR factorization on the relevant part of the $\boldsymbol{A}_{0:k+T}$, using marginal caching. Eq. 6 may encompass augmentation of the prior information, which is not detailed, for the sake of conciseness; for more details see [9].

In BSP, given an initial belief b_k and a set $\mathcal{U}$ of candidate control sequences, we wish to select the control which maximizes the expected accumulated reward, as measured with the objective function

$$J\left(b_k, u\right) \doteq \mathop{\mathbb{E}}_{\mathcal{Z}}\left[\sum_{t=1}^{T} r\left(b_{k+t-1}, \boldsymbol{u}_{k+t}\right)\right], \tag{7}$$

where $\mathcal{Z}$ is the set of observations taken while performing this control, for some reward function r. Specifically, in information-theoretic BSP, we may wish to minimize the uncertainty in the posterior belief. We use the differential entropy as an uncertainty measure, which, for a Gaussian belief b, over a state of size N, with an information matrix Λ is

$$\mathrm{H}\left(b\right) = \frac{1}{2} \cdot \ln\left[\frac{\left(2\pi e\right)^N}{|\boldsymbol{\Lambda}|}\right] = -\frac{1}{2} \cdot \left(\ln|\boldsymbol{\Lambda}| - N \cdot \ln\left(2\pi e\right)\right). \tag{8}$$

Thus, when assuming "maximum likelihood" observations (see [17]), we can define the following information-theoretic objective function:

$$J\left(b_k, u\right) \doteq \mathop{\mathbb{E}}_{\mathcal{Z}}\left[-\mathrm{H}\left(b_{k+T}\right)\right] = \ln|\boldsymbol{R}_{k+T}| - \frac{N}{2} \cdot \ln\left(2\pi e\right), \tag{9}$$

where $u \in \mathcal{U}$ is a candidate control sequence with a collective Jacobian $\boldsymbol{U}$, $\boldsymbol{R}_{k+T}$ is the posterior information root matrix, after updating $\boldsymbol{R}_k$ with $\boldsymbol{U}$, and N is the posterior state size (the number of columns in $\boldsymbol{U}$). Evidently, to calculate the value of this function, the prior information root matrix should be propagated (updated) according to the considered control sequence u. Note that the "maximum likelihood" assumption is not essential in the featured approach, but it is used for a focused and coherent discussion.

To conclude, we consider (throughout this paper) the decision problem $\mathcal{P} \doteq (b, \mathcal{U}, J)$, where b is an initial belief over the state vector $\boldsymbol{X}$; $\mathcal{U}$ is a set of candidate control actions; and J is the objective function defined in Eq. 9. To solve the problem we should select the optimal control u^*, s.t.

$$u^* = \mathop{\mathrm{argmax}}_{u \in \mathcal{U}} J\left(b, u\right). \tag{10}$$

2.2 Sparsification for Efficient BSP

As mentioned, a traditional solution to the aforementioned decision problem requires calculation of the objective function (via belief propagation) for each candidate control action. In our previous work [9], we introduced a novel method for an efficient solution of the problem: using a sparse approximation of the initial belief only while planning, without influencing the state inference. The sparsification algorithm we provided returns an approximation of a given belief, such that its information matrix and its root are sparser. The algorithm relies on a user-selected subset $\mathcal{S}$ of the state variables; this allows to generate approximations of different degrees, which trade-off sparsity and accuracy. Let us briefly analyze the algorithm steps; for a thorough analysis, please see [9].

Firstly, let us clarify our terminology, as used throughout this paper: pushing a variable forwards means to increase its index in the state vector, bringing it closer to the end of the state; accordingly, pushing a variable backwards means to decrease its index, bringing it closer to the start. We, hence, start by pushing all the variables in $\mathcal{S}$ backwards, and calculating the information root matrix $\boldsymbol{R}$ under this order. Even if the information root is given initially, this variable permutation requires to resort to the symmetric information matrix $\boldsymbol{\Lambda}$ (or Jacobians matrix), and then recalculate the factorization in the relevant order. We then remove off-diagonal elements from $\boldsymbol{R}$ in rows matching variables in $\mathcal{S}$; i.e., removing dependencies of the sparsified variables. Finally, we reorder the variables back to their original order. We showed that after the sparsification, this reordering can be done directly on the triangular root matrix, while maintaining its triangular shape, without recalculating the factorization. The sparse information matrix may be recalculated from the sparsified root, if needed. When looking at the resulting sparse information root matrix, for each sparsified variable, the algorithm replaces its corresponding entries with a single adjusted diagonal entry. Then, when applying the candidate actions, we are able to minimize the computational cost inflicted by the sparsified variables on the belief update process.

Thus far, we performed this sparsification before each planning session, and used the approximation to only efficiently solve the planning problem. This method proved to be very worthwhile. Still, at each planning session, we bared an overhead of calculating the sparsifcation from scratch, as the approximation from the previous step had been used and discarded. In this follow up work, we would like to examine the possibility to reduce this cost, by incrementally updating this sparse approximation between planning sessions, and further improve the planning performance.

3 Approach

3.1 Predictive Variable Order

A particularly interesting case of the aforementioned sparsification algorithm, is when $\mathcal{S}$ contains only variables which are *uninvolved* in any of the candidate actions. For each action u, we can identify the set of involved variables marked as $\mathcal{I}nv(u)$; these are the prior state variables which are directly updated by u, i.e. variables for which the action introduces a new constraint. In the (linearized) Gaussian least-squares representation, as presented before, this means that the column corresponding to this variable in the collective Jacobian of the action is non-zero. In a factor graph representation, this means that the action leads to connection of a new factor to this variable. The remaining variables are uninvolved in this action, and marked $\neg\mathcal{I}nv(u)$. Accordingly, the variables which are uninvolved in any of the actions $u \in \mathcal{U}$ are marked $\neg\mathcal{I}nv(\mathcal{U})$. When $\mathcal{S} = \neg\mathcal{I}nv(\mathcal{U})$, the algorithm returns an *action consistent* approximation – one which is guaranteed to not affect the action selection while planning. This paramount claim is only guaranteed due to the initial reordering of the variables in the algorithm [9]; thus, although it is the costliest step of the algorithm (due to the re-factorization), it cannot be avoided. Instead, let us avoid the final reordering of the variables, back to their original order.

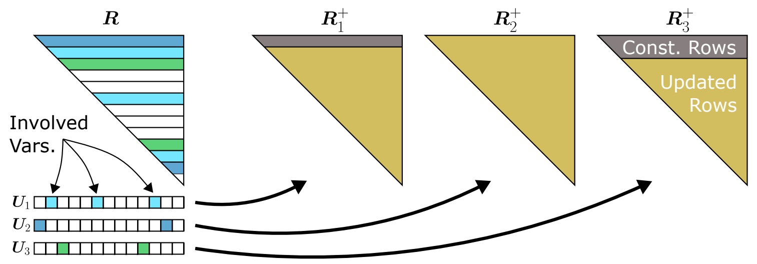
We recall that when incrementally updating a belief according to some action, the affected variables are all the state variables starting from the index of the first non-zero column of the collective Jacobian, i.e. the first involved variable; thus, the involved variables are, by definition, a subset of the affected variables. The affected variables can also include uninvolved variables which are "trapped" between the involved variables, and should only be updated due to their order. When applying the sparsification algorithm without the final reordering, the uninvolved variables $\neg\mathcal{I}nv(\mathcal{U})$ are pushed backwards before $\mathcal{I}nv(\mathcal{U})$; this makes them necessarily unaffected for all the candidate actions. In this case, as we now realize, we do not actually need to perform any sparsification. Since the uninvolved variables are not updated, their sparsification would not influence the performance anyway. We shall amplify our conclusion: by simply separating the variables according to their classification (un/involved), under the order

$$\boldsymbol{P} = \begin{bmatrix} \neg\mathcal{I}nv(\mathcal{U}) \\ \mathcal{I}nv(\mathcal{U}) \end{bmatrix}, \tag{11}$$

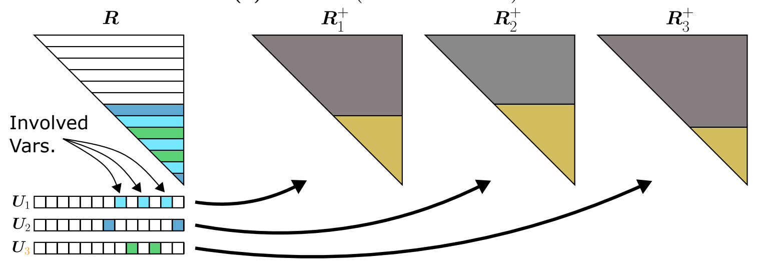
we can achieve the same computational improvement in planning previously gained with sparsification (of uninvolved variables). An illustration of this claim is brought in Fig. 1. The performance improvement results from pushing backwards the "trapped" uninvolved variables, and reducing the size of affected block for each candidate action. We may recall that updating the square root factorization has, at worst, cubical complexity in relation to the number of affected variables [10], and hence the importance of minimizing it. Since the identification of involved variables, and thus this order, is based on the predicted updates, induced by the candidate actions, we refer to $\boldsymbol{P}$ as *predictive variable order*.

To clarify, the order of variables in $\boldsymbol{P}$ matters. This separation of classes maintains the relative order of variables in each class. If variable x comes before variable y, and they both share the same classification, then even after the reordering, their relative order is kept. This means that involved variables may only be pushed forward, compared to their original position, and the number of the affected variables in each planning hypothesis may only decrease.

Optimally, for each action, we would like to update only the involved variables, and make all the uninvolved variables unaffected. However, we identify the involved variables and perform the reordering once per planning session, and not for every candidate action. Thus, for each candidate action, some of the variables in $\mathcal{I}nv(\mathcal{U})$ might not actually be involved. Still, pushing backwards even the minimum intersection of the uninvolved variables of each action (i.e. $\neg\mathcal{I}nv(\mathcal{U})$) is beneficial.



(a) Baseline (without PIVOT).



(b) With PIVOT.

Fig. 1: A conceptual demonstration of PIVOT. On the left are a *prior* information root matrix, and Jacobians of three candidate actions. Colored rows/columns represent non-zero entries, i.e. the rows/columns of the involved variables; the involved variables in each action are marked in a different color. To the right are the *posterior* root matrices matching each of the actions. Only the yellow rows should be updated, starting from the first involved variable in the action. By applying PIVOT (b) we are able to reduce the number of rows to update, in comparison to the baseline (a).

3.2 Incremental Order Update

According to our approach thus far, the sparsified belief was discarded after planning, and the selected action was applied on the original belief. We hence applied a full "batch" reordering of the root matrix, as part of the sparsification algorithm (which required a full re-factorization), before solving each planning session. In fact, when reordering the variables, we should only update the factorization starting from the index of the first reordered (affected) variable, similarly to incremental incorporation of updates. Since the classification of variables usually does not change much between planning sessions, we propose to enforce the predictive variable order on the maintained belief as well. I.e., instead of using the reordered belief only for planning, it will be kept for the following inference and planning sessions. This way, the order applied at the previous planning session is kept, and can be incrementally updated, according to the change in variable classification. When the reordering required between sessions is small, the original cost of applying the sparsification, can be reduced significantly.

Consider that in planning session k, we applied the variable order

$$\boldsymbol{P}_k = \begin{bmatrix} \neg\mathcal{I}nv(\mathcal{U}_k) \\ \mathcal{I}nv(\mathcal{U}_k) \end{bmatrix}. \tag{12}$$

We should identify the first reordered variable in planning session $k+1$, from which the incremental reordering should start. Uninvolved variables from session k which are reclassified as involved in session $k+1$, should be pushed forwards. Also, as to be explained, when applying actions between planning sessions k and $k+1$, the affected variables in those inference sessions may be shuffled. Assuming the predicted classification of variables from session k coincided with the applied action, the shuffling in inference would not affect the variables in class $\neg\mathcal{I}nv(\mathcal{U}_k)$. Alternatively, if a variable, which was classified as uninvolved in planning session k, ended up being involved in the applied action, then the order of all the variables starting from this variable may be altered (shuffled). This should not affect the position of the preceding variables. In any case, we can identify the *first m_k fixed uninvolved variables* $\boldsymbol{P}_k(1\colon m_k)$, which kept their order from planning session k.

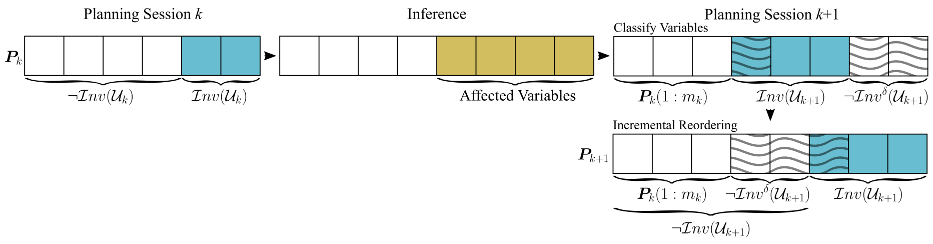

Planning Session k
Inference
Planning Session k+1
Classify Variables
$\boldsymbol{P}_k$
$\neg\mathcal{I}nv(\mathcal{U}_k)$
$\mathcal{I}nv(\mathcal{U}_k)$
Affected Variables
$\boldsymbol{P}_k(1:m_k)$
$\mathcal{I}nv(\mathcal{U}_{k+1})$
$\neg\mathcal{I}nv^{\delta}(\mathcal{U}_{k+1})$
Incremental Reordering
$\boldsymbol{P}_{k+1}$
$\boldsymbol{P}_k(1:m_k)$
$\neg\mathcal{I}nv^{\delta}(\mathcal{U}_{k+1})$
$\mathcal{I}nv(\mathcal{U}_{k+1})$
$\neg\mathcal{I}nv(\mathcal{U}_{k+1})$

Fig. 2: Incremental reordering. On the left is the state under the predictive variable order, as applied before planning session k; the variables in blue are classified as involved. In the middle is the state after performing the selected action. Only the variables in yellow are affected by the inference updates (including new variables). In planning session $k+1$, on the right, we reclassify the variables. Variables with a new classification are marked with stripes. The first m_k variables remain uninvolved, and we can incrementally reorder the variables to the desired predictive order.

Besides these variables, we use $\neg\mathcal{I}nv^{\delta}(\mathcal{U}_{k+1})$ to mark all the remaining uninvolved variables at session $k+1$. This class would contain uninvolved variables which were shuffled forward in inference, and new uninvolved variables, whether they were added since the previous planning session, or variables which changed their class from planning session k. By definition, all these variables should come after the fixed uninvolved variables from the previous session $\boldsymbol{P}_k(1\colon m_k)$. The applied order in session $k+1$ would then be

$$\boldsymbol{P}_{k+1} = \begin{bmatrix} \boldsymbol{P}_k(1\colon m_k) \\ \neg\mathcal{I}nv^{\delta}(\mathcal{U}_{k+1}) \\ \mathcal{I}nv(\mathcal{U}_{k+1}) \end{bmatrix}, \tag{13}$$

and the first variable to reorder would be (at least) $m_k + 1$. This analysis is demonstrated in Fig. 2. This is the most general case, with no assumptions on the reordering applied between planning sessions. Notably, if at the time of reordering, there are no variables from class $\mathcal{I}nv(\mathcal{U}_{k+1})$ "trapped" between variables from classes $\boldsymbol{P}_k(1\colon m_k)$ and $-\mathcal{I}nv^{\delta}(\mathcal{U}_{k+1})$, or alternatively, if $\left|-\mathcal{I}nv^{\delta}(\mathcal{U}_{k+1})\right| = 0$, then no reordering is required.

To conclude our suggested approach, before every planning session we shall reclassify the variables according to the predicted candidate actions, and perform an incremental reordering procedure on the current belief accordingly. The planning problem should be then solved regularly, e.g. as specified in Section 2.1. We thus refer to the approach as *Predictive Incremental Variable Ordering Tactic (PIVOT).* The tactic is intended to benefit the efficiency of the planning, which is a much harder problem than state inference. In planning we should account to multiple candidate realizations, with (usually) longer horizons. Still, since PIVOT is applied on the maintained belief, its influence on the inference process needs to be examined.

3.3 Influence on State Inference

Before we can examine the effects of PIVOT on the state inference, we shall understand what the default baseline tactic is. Variable ordering in the information matrix can have a dramatic effect on the number of non-zeros (i.e. fill-in) in its triangular square root matrix. As explained, to solve the state inference problem, we should maintain this root matrix, achieved via Cholesky or QR factorization. Fill-reducing variable ordering tactics are very commonly applied in the solution of the state inference problem. Although finding the optimal variable order is NP-complete, good heuristics exist; COLAMD is a prominent one [4].

In sequential state inference, we can optimize the order of all state variables, as a periodical "batch" procedure, which is followed by a recalculation of the entire root matrix under the new order (e.g. [14]). Alternatively, in incremental belief updates, we can optimize the order of only the affected variables, as a part of the refactorization of the affected block (as in [11,13]). Incremental application of fill-reducing ordering tactics tend to be inferior to their batch counterpart in the resulting fill-in, but are more efficient.

We notice that recent algorithms such as iSAM2 [13], as part of the inference update, also constrain the variables involved in the applied action to the end of the state, besides applying COLAMD. This does not optimize the current inference session – on the contrary, it leads to a sacrifice in fill-in – but does contribute to the following inference sessions. If the variables are likely to be updated again, then the affected part would be smaller. In their experiments the authors demonstrated that this claim is well worthwhile. As the current state of the art, we will refer to this tactic, constrained incremental COLAMD, as the baseline to our comparison. Accordingly, PIVOT affects the state inference in two main aspects:

The first – loop closing. As explained, incorporating updates to variables from the past (i.e. loop closures) is expensive, as these affect numerous variables. In our approach, we conduct a variable reordering procedure before every planning session, which pushes forward *predicted* loop closing variables, according to any of the candidate actions. Unlike the constrained baseline approach, we push forward these variables *before* the loop closures occur, as a part of the planning process. Thus, we can reduce the cost of future state inference iterations containing big loop closures, when they actually occurs. We should note that since we identify the involved variables based on the entire set of candidate actions, we constrain to the end of the state more variables than we would have for a single action. Still, if we maintain the relative order of the involved variables, then loop closures can only move forward. When ignoring the planning problem, there is no point in actively bringing forward loop-closing variables, as the cost of this 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.

Another aspect to consider – fill-in. Applying PIVOT is intended to increase planning efficiency, in the ways indicated, and is not inherently fill-reducing. Like the baseline, constraining the involved variables forward may lead to suboptimal fill-in and a denser information root matrix. Fill-in in the root matrix is usually correlated with the factorization performance – more fill-in indicates (usually) a slower calculation. However, when examining incremental factorization updates, we should only look at the number of non-zero elements only in the recalculated affected block. Surely, the affected block becomes smaller when applying PIVOT; hence, despite the added density, the number of recalculated non-zero elements shall remain balanced, and the update performance should not suffer. We demonstrate this claim in the following experimental results.

Besides that, added fill-in in the root matrix might increase the cost of back-substitution, which is solved when updating the state MAP estimate. Although we expect some sacrifice in back-substitution cost, the overall gains in performance should prevail. 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 estimation of the last pose, which is usually at the end of the state, can be achieved almost immediately. Regardless, in planning, assuming "maximum likelihood" observations, there is no need to perform back-substitution, as the posterior state estimate does not change [17,12].

4 Experimental Analysis

4.1 The Scenario

As a proof of concept, we applied PIVOT in an active-SLAM simulation. In this scenario, an agent autonomously navigates to a specified goal in a mostly-unknown environment, in which random landmarks are scattered. The agent is equipped with a radar-type sensor, providing range-bearing observations of surrounding landmarks. To ignore collisions, the agent and landmarks are considered mass-less. Largely, the full map is unknown to the agent, and it is inferred by it during the navigation. Still, the agent begins its task with a prior belief over some landmarks in known areas. This scenario can also be viewed as if the agent had previously performed an exploration task, and had already mapped parts of the environment.

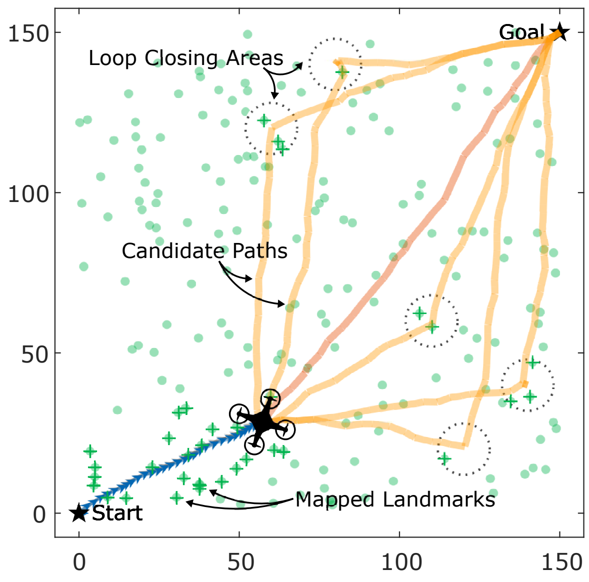


Fig. 3: The active-SLAM simulation during a planning session. The agent's estimated trajectory is in blue; underneath it, in grey, is the ground-truth. The start point is $(0\text{m}, 0\text{m})$, and the navigation goal is $(150\text{m}, 150\text{m})$. Ground-truth position of all landmarks is in light green; darker green crosses indicate estimation of observed landmarks. Areas with prior knowledge are marked with circles. Candidate paths to the goal in the current planning session are in orange – dark tone for a direct path, and light tone for loop-closing paths. The maximal step size between poses, marked with arrowheads, is 5m. Sensing range is 15m.

After passing a set distance, a new pose is added to the state, and the sensor scans the environment. New landmarks are added to the end of the state, and constraints are added between the new pose and the observed landmarks. The data association is assumed to be known. Motion constraints are also created between every two consecutive poses. Both the observation and motion contain some Gaussian noise. After adding the new motion/observation constraints, the agent updates its belief over the state by incrementally updating the square root information matrix (i.e. a state inference session). Overall, the agent's state $\boldsymbol{X}_k$ consists of the entire executed trajectory and positions of observed landmarks (i.e. full-SLAM). Each of the poses consists of three variables, representing the position and orientation; each landmark is represented via two position variables. Our approach is highly relevant in this case; as the agent progresses towards the goal, past poses and landmarks are expected to become uninvolved.

From its starting point, the agent begins a planning session by generating a set $\mathcal{U}$ of candidate trajectories to the goal: one direct trajectory, and several indirect trajectories which pass through the known areas on the map, in order to identify loop closure and reduce uncertainty. The trajectories are sampled using the Probabilistic RoadMmap (PRM) algorithm [15]. Each candidate trajectory matches a certain control sequence, and is translated to a series of constraints and variables to be added to the prior belief. Loop closure constraints are added between future poses and landmarks from the past, according to their estimated location, i.e. where we expect to add them when executing this trajectory. The scenario can be viewed in Fig. 3.

For planning, we use the following objective function, which balances the two sub-goals – minimizing both the length of the trajectory, and the uncertainty of the state, by preferring a more informative trajectory:

$$\hat{J}(b, u) \doteq \omega_1 \cdot J(b, u) + \omega_2 \cdot \mathrm{length}(u), \tag{14}$$

where ω_1, $\omega_2 \in \mathbb{R}$ are scalar weights and $J(b, u)$ measures the posterior entropy, as defined in Eq. 9. As explained, calculating $J(b, u)$ requires propagation of the belief's root matrix; the expected trajectory's length, however, does not depend on it, under "maximum likelihood" assumption. The agent partially executes the selected trajectory (i.e. a set number of state inference sessions), and then conducts a new planning session, with newly generated trajectories. This re-planning is important in order to account for the previously-unknown state development. The agent repeats the planning-execution cycle until reaching its goal.

To evaluate our method, we maintained in parallel two versions of the belief, under different ordering tactics: the baseline (as explained in Section 3.3), and PIVOT; we performed all belief updates on both versions. For each version, we measured and compared the total state inference and planning times. For each inference session, we measured the time of updating the information root matrix; for each planning session, we measured the matrix update time for each candidate action, along with the time of reordering the variables in the root matrix, when applying PIVOT. For the sake of comparison, in each planning session we also measured the reordering time when applying the predictive ordering from scratch on the original baseline belief (i.e. non-incrementally). This configuration represents non-incremental AC sparsification (see Section 2.2). We will refer to this configuration as "batch reordering". We recall that reordering the variables does not affect the accuracy of the solution.

The belief was maintained using the MATLAB interface of the GTSAM library [5]. We still wished to fairly measure the reordering and update time in the matrix form, without the specific framework overhead, nor accounting for re-linearization. Therefore, at each state inference and planning sessions, we extracted the prior root matrix, and the relevant linearized Jacobians, and performed the matrix updates manually, for timing.

4.2 Results

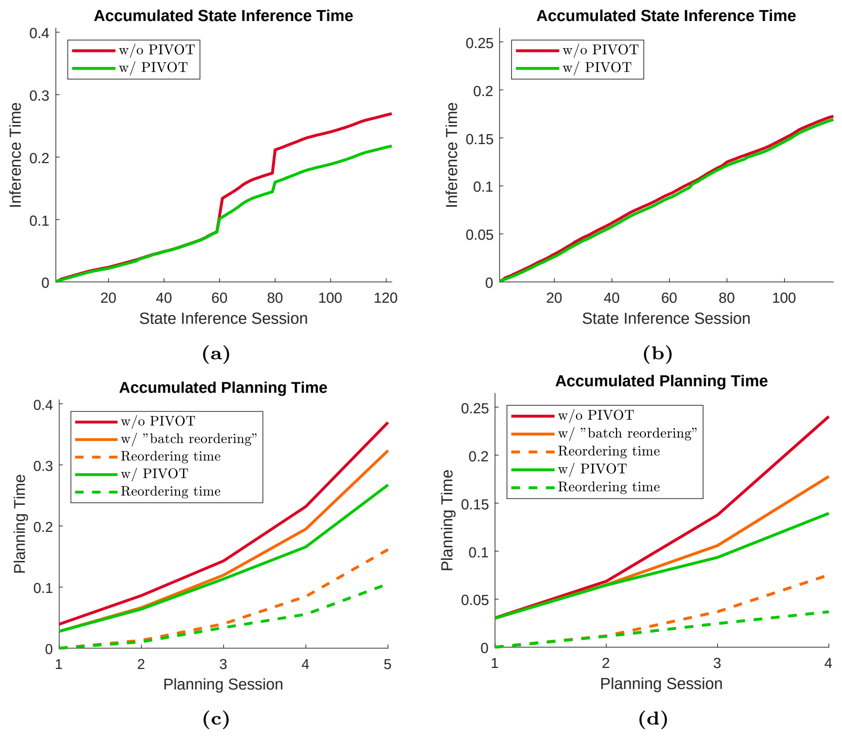


Fig. 4: Accumulated planning and inference times in two outcomes of the same scenario: (1) when the selected path includes loop-closing in the known areas – represented by the two figures on the left (a and c); and (2) when the agent headed directly to the goal – represented by the two figures on the right (b and d). For each inference session, we measured the time of updating the information root matrix; for each planning session, we measured the matrix update time for each candidate action, along with the time of reordering the variables in the root matrix. In both cases, agent performed at most 30 inference steps before re-planning. The x axes overlap between the graphs of each outcome. Surely, lower computation times are better.

Fig. 4 presents the accumulated planning and inference times. We present two possible outcomes of *the same* scenario, according to different assignment of weights in the objective function – first when the selected path included loop-closing in the known areas, and second when the agent headed directly to the goal. In both cases, the agent performed at most 30 inference steps before re-planning. Since the trajectory in outcome 1 is longer, it resulted in 5 planning sessions, compared to 4 in outcome 2. For this reason, the accumulated planning and inference times in outcome 1 are higher than in outcome 2. We therefore only care for the relative performance. The numerical values from Fig. 4 are given in Table 1.

		Loop-closing trajectory			**Direct trajectory**		
		Baseline	"Batch"	PIVOT	Baseline	"Batch"	PIVOT
Inference	Update	0.27	0.27	0.22	0.172	0.172	0.169
Planning	Update	0.37	0.16	0.16	0.24	0.10	0.10
	Reordering	0	0.16	0.10	0	0.07	0.03
Total time		0.64	0.59	0.48	0.41	0.35	0.31

Table 1: Numerical summary for the accumulated inference and planning times presented in Fig. 4.

For both outcomes, using PIVOT resulted in a substantial 25% improvement in overall performance, without any sacrifice in accuracy. However, the detailed results differ between the two outcomes. In outcome 1, we faced two large loop-closures, around pose #60 and pose #80, as can be seen in Fig. 4a. We recall that the baseline ordering pushes forwards the loop-closing variables only *after* they are met. Applying PIVOT pushed the loop-closing variables forwards *before* meeting them, in the preceding planning sessions; this resulted in a more efficient loop-closing in inference compared to the baseline. After facing the loop-closures, the baseline ordering coincided with PIVOT; thus, in the following planning sessions, when considering these loop-closure again, the update cost was similar for both PIVOT and the baseline. Applying PIVOT was still beneficial to planning when examining the other loop-closure, as the loop-closing variables which had not been met, remained backwards under the baseline ordering. In outcome 2, we faced no major loop closures, thus PIVOT did not benefit the baseline inference in the ways we described before, and inference time was kept (almost) the same. Also, under the baseline ordering, all the loop-closing variables did not move forwards, and loop-closing remained expensive in all planning sessions; thus, the relative performance improvement in planning is higher in outcome 2.

The performance using "batch reordering" falls between PIVOT and the baseline, as expected. Since the planning is conducted with the predictive order, the update time in planning is the same as in PIVOT. But since this order is discarded and not applied on the maintained state, the inference time remains the same as in the baseline. We can see that applying the order incrementally significantly improves the reordering time, as we originally intended, while the inference performance remains the same or better.

Fig. 5 shows the difference in fill-in in the two versions of the information root matrix at the end of the trajectory. The constrained order leads to a slight increase in the number of non-zero elements.

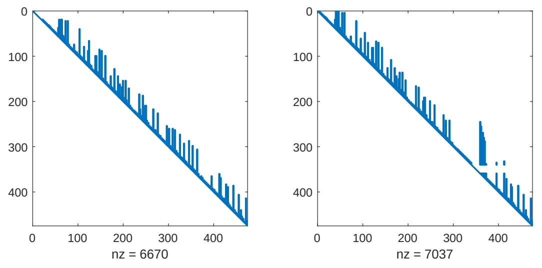


Fig. 5: A comparison of the sparsity pattern of two versions of the information root matrix, after reaching the goal through the direct trajectory: baseline (left), and PIVOT (right). The constrained order leads to a slight increase in the number of non-zeros.

5 Conclusion

We presented PIVOT, Predictive Incremental Variable Ordering Tactic, a novel paradigm to improve the efficiency of BSP, with no sacrifice in accuracy. We started by explaining the relation to our previous line of work regarding belief sparsification in the context of planning. We suggested to apply a predictive variable order before the beginning of the planning session; this order pushes backwards all the variables which are classified as "uninvolved", based on the set of candidate actions. This order helps minimizing the cost of belief propagation, based on these actions; specifically, the cost of updating the information root matrix. This order can be incrementally updated between planning sessions, based on the change in the variable classes.

We also saw that as a by-product of applying the tactic, we are able to cut down on the cost of loop-closing in inference. Still, the approach is only relevant in the context of planning, as the predictive classification of variables relies on the candidate actions. Also, precursory reordering only makes sense when applying multiple actions on the initial belief; otherwise, the reordering can be combined with the update itself.

To demonstrate its benefit, we applied PIVOT in an active-SLAM simulation. In different scenarios we were able to significantly reduce computation time of both the planning and inference. The improvement achieved by PIVOT depends on multiple various parameters; examining the influence of all of them is well beyond the scope of this paper. These include, for example, the number of inference sessions per planning sessions, belief structure (e.g. full-SLAM vs. pose-SLAM), ratio of uninvolved variables, number of candidate actions, etc.. Examining these parameters, as well as solutions to the added fill-in the constrained order might invoke, are left for future work.

References

1. Pratik Agarwal and Edwin Olson. Variable reordering strategies for slam. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3844–

3850. IEEE, 2012.

2. Blai Bonet and Héctor Geffner. Planning with incomplete information as heuristic search in belief space. In *Intl. Conf. on Artificial Intelligence Planning Systems*, pages 52–61. AAAI Press, 2000.
3. Stephen M Chaves and Ryan M Eustice. Efficient planning with the Bayes tree for active SLAM. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4664–4671. IEEE, 2016.
4. T.A. Davis, J.R. Gilbert, S.I. Larimore, and E.G. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376, 2004.
5. F. Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology, September 2012.
6. F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, 25(12):1181–1203, Dec 2006.
7. K. Elimelech and V. Indelman. Consistent sparsification for efficient decision making under uncertainty in high dimensional state spaces. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2017.
8. K. Elimelech and V. Indelman. Scalable sparsification for efficient decision making under uncertainty in high dimensional state spaces. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, September 2017.
9. K. Elimelech and V. Indelman. Efficient decision making and belief space planning using sparse approximations. *arXiv preprint arXiv:1909.00885*, 2018.
10. Günther Hämmerlin and Karl-Heinz Hoffmann. *Numerical mathematics*. Springer Science & Business Media, 2012.
11. Viorela Ila, Lukas Polok, Marek Solony, and Pavel Svoboda. SLAM++ - a highly efficient and temporally scalable incremental slam framework. *Intl. J. of Robotics Research*, 36(2):210–230, 2017.
12. V. Indelman, L. Carlone, and F. Dellaert. 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, 2015.
13. M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research*, 31:217–236, Feb 2012.
14. M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378, Dec 2008.
15. L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, 1996.
16. S. Patil, G. Kahn, M. Laskey, J. Schulman, K. Goldberg, and P. Abbeel. Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation. In *Intl. Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 515–533, 2014.
17. R. Platt, R. Tedrake, L.P. Kaelbling, and T. Lozano-Pérez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems (RSS)*, pages 587–593, Zaragoza, Spain, 2010.
18. J. M. Porta, N. Vlassis, M. T. Spaan, and P. Poupart. Point-based value iteration for continuous pomdps. *J. of Machine Learning Research*, 7:2329–2367, 2006.
19. S. Prentice and N. Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *Intl. J. of Robotics Research*, 28(11-12):1448–1465, 2009.