Consensus Maximization Tree Search Revisited

Zhipeng Cai
The University of Adelaide

Tat-Jun Chin
The University of Adelaide

Vladlen Koltun
Intel Labs

Abstract

Consensus maximization is widely used for robust fitting in computer vision. However, solving it exactly, i.e., finding the globally optimal solution, is intractable. A tree search, which has been shown to be fixed-parameter tractable, is one of the most efficient exact methods, though it is still limited to small inputs. We make two key contributions towards improving A* tree search. First, we show that the consensus maximization tree structure used previously actually contains paths that connect nodes at both adjacent and non-adjacent levels. Crucially, paths connecting non-adjacent levels are redundant for tree search, but they were not avoided previously. We propose a new acceleration strategy that avoids such redundant paths. In the second contribution, we show that the existing branch pruning technique also deteriorates quickly with the problem dimension. We then propose a new branch pruning technique that is less dimension-sensitive to address this issue. Experiments show that both new techniques can significantly accelerate A* tree search, making it reasonably efficient on inputs that were previously out of reach. Demo code is available at* `https://github.com/ZhipengCai/MaxConTreeSearch`.

1. Introduction

The prevalence of outliers makes robust model fitting crucial in many computer vision applications. One of the most popular robust fitting criteria is *consensus maximization*, whereby, given outlier-contaminated data $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^N$, we seek the model $\boldsymbol{\theta} \in \mathbb{R}^d$ that is consistent with the largest subset of the data. Formally, we solve

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad c(\boldsymbol{\theta}|\mathcal{S}) = \sum_{i=1}^{N} \mathbb{I}\{r(\boldsymbol{\theta}|\mathbf{s}_i) \leq \epsilon\}, \qquad (1)$$

where $c(\boldsymbol{\theta}|\mathcal{S})$ is called the *consensus* of $\boldsymbol{\theta}$. The 0/1 valued indicator function $\mathbb{I}\{\cdot\}$ returns 1 only when $\mathbf{s}_i$ is consistent with $\boldsymbol{\theta}$, which happens when the residual $r(\boldsymbol{\theta}|\mathbf{s}_i) \leq \epsilon$. The form of $r(\boldsymbol{\theta}|\mathbf{s}_i)$ will be defined later in Sec. 2. Constant ϵ is the predefined inlier threshold, and d is called the "problem dimension". Given the optimal solution $\boldsymbol{\theta}^*$ of (1), $\mathbf{s}_i$ is an inlier if $r(\boldsymbol{\theta}^*|\mathbf{s}_i) \leq \epsilon$ and an outlier otherwise.

Consensus maximization is NP-hard [4], hence, sub-optimal but efficient methods are generally more practical. Arguably the most prevalent methods of this type are RANSAC [11] and its variants [8, 26, 7, 24], which iteratively fit models on randomly sampled (minimal) data subsets and return the model with the highest consensus. However, their inherent randomness makes these methods often distant from optimal and sometimes unstable. To address this problem, deterministic optimization techniques [23, 14, 2] have been proposed, which, with good initializations, usually outperform RANSAC variants. Nonetheless, a good initial solution is not always easy to find. Hence, these methods may still return unsatisfactory results.

The weaknesses of sub-optimal methods motivate researchers to investigate globally optimal methods; however, so far they are effective on only small input sizes (small d, N and/or number of outliers o). One of the most efficient exact methods is tree search [15, 5, 6] (others surveyed later in Sec. 1.1), which fits (1) into the framework of the LP-type methods [25, 18]. By using heuristics to guide the tree search and conduct branch pruning, A* tree search [5, 6] has been demonstrated to be much faster than Breadth-First Search (BFS) and other types of globally optimal algorithms. In fact, tree search is provably fixed-parameter tractable (FPT) [4]. Nevertheless, as demonstrated in the experiment of [6] and later ours, A* tree search can be highly inefficient for challenging data with moderate d (≥ 6) and o (≥ 10).

Our contributions. In this work, we analyze reasons behind the inefficiency of A* tree search and develop improvements to the algorithm. Specifically:

- We demonstrate that the previous tree search algorithm does not avoid all redundant paths, namely, paths that connect nodes from non-adjacent levels. Based on this observation, a new acceleration strategy is proposed, which can avoid such non-adjacent (and redundant) paths.
- We show that the branch pruning technique in [6] is not always effective and may sometimes slow down the tree search due to its sensitivity to d. To address this problem, we propose a branch pruning technique that is less dimension-sensitive and hence much more effective.

Experiments demonstrate the significant acceleration achievable using our new techniques (3 orders of magnitude

faster on challenging data). Our work represents significant progress towards making globally optimal consensus maximization practical on real data.

1.1. Related Work

Besides tree search, other types of globally optimal methods include branch-and-bound (BnB) [16, 28, 22], whose exhaustive search is done by testing all possible $\boldsymbol{\theta}$. However, the time complexity of BnB is exponential in the size of the parameter space, which is often large. Moreover, the bounding function of BnB is problem-dependent and not always trivial to construct. Another type of methods [20, 9] enumerate and fit models on all possible bases, where each basis is a data subset of size p, where $p \ll N$ and p is usually slightly larger than d, e.g., $p = d + 1$. The number of all possible bases is $\binom{N}{p}$, which scales poorly with N and d. Besides differences in actual runtime, what distinguishes tree search from the other two types of methods is that tree search is FPT [4]: its worst case runtime is exponential in d and o, but polynomial in N.

2. Consensus maximization tree search

We first review several concepts that are relevant to consensus maximization tree search.

2.1. Application range

Tree search requires the residual $r(\boldsymbol{\theta}|\mathbf{s}_i)$ to be *pseudo-convex* [6]. A simple example is the linear regression residual

$$r(\boldsymbol{\theta}|\mathbf{s}_i) = |\mathbf{a}_i^T\boldsymbol{\theta} - b_i|, \tag{2}$$

where each datum $\mathbf{s}_i = \{\mathbf{a}_i, b_i\}$, $\mathbf{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$. Another example is the residual used in common multiview geometry problems [21, 2], which are of the form

$$r(\boldsymbol{\theta}|\mathbf{s}_i) = \frac{\left\|\mathbf{A}_i^T\boldsymbol{\theta} - \mathbf{b}_i\right\|_p}{\mathbf{c}_i^T\boldsymbol{\theta} - d_i}, \tag{3}$$

where each datum $\mathbf{s}_i = \{\mathbf{A}_i, \mathbf{b}_i, \mathbf{c}_i, d_i\}$, $\mathbf{A}_i \in \mathbb{R}^{d\times m}$, $\mathbf{b}_i \in \mathbb{R}^m$, $\mathbf{c}_i \in \mathbb{R}^d$ and $d_i \in \mathbb{R}$. Usually, p is 1, 2 or ∞.

2.2. LP-type problem

The tree search algorithm for (1) is constructed by solving a series of minimax problems, which are of the form

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \ \max_{i\in\mathcal{S}^1} \ r(\boldsymbol{\theta}|\mathbf{s}_i). \tag{4}$$

Problem (4) minimizes the maximum residual for all data in $\mathcal{S}^1$, which is an arbitrary subset of $\mathcal{S}$. For convenience, we define $f(\mathcal{S}^1)$ as the minimum objective value of (4) computed on data $\mathcal{S}^1$, and $\boldsymbol{\theta}(\mathcal{S}^1)$ as the (exact) minimizer.

Throughout the paper, we will assume that $r(\cdot)$ is pseudo-convex and $\mathcal{S}$ is non-degenerate (otherwise infinitestimal perturbations can be applied to remove degeneracy [18, 6]). Under this assumption, problem (4) has a unique optimal solution and can be solved efficiently with standard solvers [10]. Furthermore, (4) is provably an LP-type problem [25, 1, 10], which is a generalization of the linear programming (LP) problem. An LP-type problem has the following properties:

Property 1 (Monotonicity). *For every two sets $\mathcal{S}^1 \subseteq \mathcal{S}^2 \subseteq \mathcal{S}$, $f(\mathcal{S}^1) \leq f(\mathcal{S}^2) \leq f(\mathcal{S})$.*

Property 2 (Locality). *For every two sets $\mathcal{S}^1 \subseteq \mathcal{S}^2 \subseteq \mathcal{S}$ and every $\mathbf{s}_i \in \mathcal{S}$, $f(\mathcal{S}^1) = f(\mathcal{S}^2) = f(\mathcal{S}^2 \cup \{\mathbf{s}_i\}) \Rightarrow f(\mathcal{S}^1) = f(\mathcal{S}^1 \cup \{\mathbf{s}_i\})$.*

With the above properties, the concept of *basis*, which is essential for tree search, can be defined.

Definition 1 (Basis). *A basis $\mathcal{B}$ in $\mathcal{S}$ is a subset of $\mathcal{S}$ such that for every $\mathcal{B}' \subset \mathcal{B}$, $f(\mathcal{B}') < f(\mathcal{B})$.*

For an LP-type problem (4) with pseudo-convex residuals, the maximum size of a basis, which we call *combinatorial dimension*, is $d + 1$.

Definition 2 (Violation set, level and coverage). *The violation set of a basis $\mathcal{B}$ is defined as $\mathcal{V}(\mathcal{B}) = \{\mathbf{s}_i \in \mathcal{S} | \ r(\boldsymbol{\theta}(\mathcal{B})|\mathbf{s}_i) > f(\mathcal{B})\}$. We call $l(\mathcal{B}) = |\mathcal{V}(\mathcal{B})|$ the level of $\mathcal{B}$ and $\mathcal{C}(\mathcal{B}) = \mathcal{S}\backslash\mathcal{V}(\mathcal{B})$ the coverage of $\mathcal{B}$.*

By the above definition,

$$c(\boldsymbol{\theta}(\mathcal{B})|\mathcal{S}) = |\mathcal{S}| - l(\mathcal{B}). \tag{5}$$

An important property of LP-type problems is that solving (4) on $\mathcal{C}(\mathcal{B})$ and $\mathcal{B}$ return the same solution.

Definition 3 (Support set). *The level-0 basis for $\mathcal{S}$ is called the support set of $\mathcal{S}$, which we represent as $\tau(\mathcal{S})$.*

Assume we know the maximal inlier set $\mathcal{I}$ for (1), where $|\mathcal{I}| = c(\boldsymbol{\theta}^*|\mathcal{S})$. Define $\mathcal{B}^* = \tau(\mathcal{I})$ as the support set of $\mathcal{I}$; $\mathcal{B}^*$ can be obtained by solving (4) on $\mathcal{I}$. Then, $l(\mathcal{B}^*)$ is the size of the minimal outlier set. Our target problem (1) can then be recast as finding the optimal basis

$$\mathcal{B}^* = \underset{\mathcal{B}\subseteq\mathcal{S}}{\text{argmin}} \ l(\mathcal{B}), \ \text{s.t.} \ f(\mathcal{B}) \leq \epsilon, \tag{6}$$

and $\boldsymbol{\theta}(\mathcal{B}^*)$ is the maximizer of (1). Intuitively, $\mathcal{B}^*$ is the lowest level basis that is feasible, where a basis $\mathcal{B}$ is called feasible if $f(\mathcal{B}) \leq \epsilon$.

2.3. A* tree search algorithm

Matoušek [18] showed that the set of bases for an LP-type problem can be arranged in a tree, where the root node is $\tau(\mathcal{S})$, and the level occupied by a node $\mathcal{B}$ on the tree is $l(\mathcal{B}) = |\mathcal{V}(\mathcal{B})|$. Another key insight is that there exists a path from $\tau(\mathcal{S})$ to any higher level basis, where a path is formed by a sequence of *adjacent bases*, defined as follows.

Algorithm 1 A* tree search of Chin et al. [6] for (6)

Require: $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^N$, threshold ϵ.
1: Insert $\mathcal{B} = \tau(\mathcal{S})$ with priority $e(\mathcal{B})$ into queue q.
2: Initialize hash table T to NULL.
3: **while** q is not empty **do**
4: Retrieve from q the $\mathcal{B}$ with the lowest $e(\mathcal{B})$.
5: **if** $f(\mathcal{B}) \leq \epsilon$ **then**
6: return $\mathcal{B}^* = \mathcal{B}$.
7: **end if**
8: $\mathcal{B}_r \leftarrow$ Attempt to reduce $\mathcal{B}$ by TOD method.
9: **for** each $\mathbf{s} \in \mathcal{B}_r$ **do**
10: **if** indices of $\mathcal{V}(\mathcal{B}) \cup \{\mathbf{s}\}$ do not exist in T **then**
11: Hash indices of $\mathcal{V}(\mathcal{B}) \cup \{\mathbf{s}\}$ into T.
12: $\mathcal{B}' \leftarrow \tau(\mathcal{C}(\mathcal{B}) \backslash \{\mathbf{s}\})$.
13: Insert $\mathcal{B}'$ with priority $e(\mathcal{B}')$ into q.
14: **end if**
15: **end for**
16: **end while**
17: Return error (no inlier set of size greater than p).

Definition 4 (**Basis adjacency**). *Two bases $\mathcal{B}'$ and $\mathcal{B}$ are adjacent if $\mathcal{V}(\mathcal{B}') = \mathcal{V}(\mathcal{B}) \cup \{\mathbf{s}_i\}$ for some $\mathbf{s}_i \in \mathcal{B}$.*

Intuitively, $\mathcal{B}'$ is a direct child of $\mathcal{B}$ in the tree. We say that we "follow the edge" from $\mathcal{B}$ to $\mathcal{B}'$ when we compute $\tau(\mathcal{C}(\mathcal{B}) \backslash \{\mathbf{s}_i\})$. Chin et al. [6] solve (6) by searching the tree structure using the A* shortest path finding technique (Algorithm 1). Given input data $\mathcal{S}$, A* tree search starts from the root node $\tau(\mathcal{S})$ and iteratively expands the tree until $\mathcal{B}^*$ is found. The queue q stores all unexpanded tree nodes. And in each iteration, a basis $\mathcal{B}$ with the lowest *evaluation* value $e(\mathcal{B})$ is expanded. The expansion follows the basis adjacency, which computes $\tau(\mathcal{C}(\mathcal{B}) \backslash \{\mathbf{s}\})$ for all $\mathbf{s} \in \mathcal{B}$ (Line 12 in Algorithm 1).

The evaluation value is defined as

$$e(\mathcal{B}) = l(\mathcal{B}) + h(\mathcal{B}), \tag{7}$$

where $h(\mathcal{B})$ is a *heuristic* which estimates the number of outliers in $\mathcal{C}(\mathcal{B})$. A* search uses only *admissible* heuristics.

Definition 5 (**Admissibility**). *A heuristic h is admissible if $h(\mathcal{B}) \geq 0$ and $h(\mathcal{B}) \leq h^*(\mathcal{B})$, where $h^*(\mathcal{B})$ is the minimum number of data that must removed from $\mathcal{C}(\mathcal{B})$ to make the remaining data feasible.*

Note that setting $e(\mathcal{B}) = l(\mathcal{B})$ (i.e., $h(\mathcal{B}) = 0$) for all $\mathcal{B}$ reduces A* search to breadth-first search (BFS). With an admissible heuristic, A* search is guaranteed to always find $\mathcal{B}^*$ before other sub-optimal feasible bases (see [6] for the proof). Algorithm 2 describes the heuristic h_{ins} used in [6].

Intuitively, the algorithm for h_{ins} removes a sequence of bases in the first round of iteration until a feasible subset $\mathcal{F} \subseteq \mathcal{C}(\mathcal{B})$ is found. After that, the algorithm iteratively inserts each removed basis point $\mathbf{s}$ back into $\mathcal{F}$. If the insertion

Algorithm 2 Admissible heuristic h_{ins} for A* tree search

Require: $\mathcal{B}$
1: If $f(\mathcal{B}) \leq \epsilon$, return 0.
2: $\mathcal{O} \leftarrow \emptyset$.
3: **while** $f(\mathcal{B}) > \epsilon$ **do**
4: $\mathcal{O} \leftarrow \mathcal{O} \cup \mathcal{B}$, $\mathcal{B} \leftarrow \tau(\mathcal{C}(\mathcal{B}) \backslash \mathcal{B})$.
5: **end while**
6: $h_{ins} \leftarrow 0$, $\mathcal{F} \leftarrow \mathcal{C}(\mathcal{B})$.
7: **for** each $\mathcal{B} \in \mathcal{O}$ **do**
8: **for** each $\mathbf{s} \in \mathcal{B}$ **do**
9: $\mathcal{B}' \leftarrow \tau(\mathcal{F} \cup \{\mathbf{s}\})$.
10: **if** $f(\mathcal{B}') \leq \epsilon$ **then**
11: $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathbf{s}\}$.
12: **else**
13: $h_{ins} \leftarrow h_{ins} + 1$, $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathbf{s}\} \backslash \mathcal{B}'$.
14: **end if**
15: **end for**
16: **end forreturn** h_{ins}.

of $\mathbf{s}$ makes $\mathcal{F}$ infeasible, $\tau(\mathcal{F} \cup \{\mathbf{s}\})$ is removed from the expanded $\mathcal{F}$ and the heuristic value h_{ins} is increased by 1.

The admissibility of h_{ins} is proved in [6, Theorem 4]. In brief, denote $\mathcal{F}^*$ as the largest feasible subset of $\mathcal{C}(\mathcal{B})$. If $\mathcal{F} \cup \{\mathbf{s}\}$ is infeasible, $\tau(\mathcal{F} \cup \{\mathbf{s}\})$ must contain at least one point in $\mathcal{F}^*$. Since we only add 1 to h_{ins} when this happens, then $h^*(\mathcal{B}) \geq h_{ins}(\mathcal{B})$.

2.4. Avoiding redundant node expansions

Algorithm 1 employs two strategies to avoid redundant node expansions. In Line 8, before expanding $\mathcal{B}$, a fast heuristic called True Outlier Detection (TOD) [6] is used to attempt to identify and remove true outliers from $\mathcal{B}$ (more details in Sec. 4), which has the potential to reduce the size of the branch starting from $\mathcal{B}$. In Line 10, a repeated basis check heuristic is performed to prevent bases that have been explored previously to be considered again (details in Sec. 3).

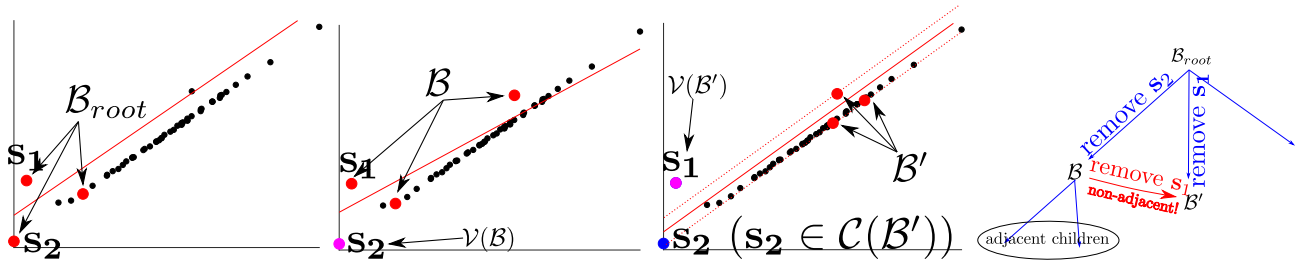
Our main contributions are two new strategies that improve upon the original methods above, as we will describe in Secs. 3 and 4. In each of the sections, we will first carefully analyze the weaknesses of the existing strategies. Sec. 5 will then put our new strategies in an overall algorithm. Sec. 6 presents the results.

3. Non-adjacent path avoidance

Recall Definition 4 on adjacency: for $\mathcal{B}$ and $\mathcal{B}'$ to be adjacent, their violation sets $\mathcal{V}(\mathcal{B})$ and $\mathcal{V}(\mathcal{B}')$ must differ by one point; in other words, it must hold that

$$|l(\mathcal{B}') - l(\mathcal{B})| = 1. \tag{8}$$

Given a $\mathcal{B}$, Line 12 in Algorithm 1 generates an adjacent "child" basis of $\mathcal{B}$ by removing a point $\mathbf{s}$ from $\mathcal{B}$ and solving



(a) Root node $\mathcal{B}_{root}$. (b) Level-1 node $\mathcal{B}$. (c) Level-1 node $\mathcal{B}$. $\mathbf{s}_2 \in \mathcal{C}(\mathcal{B})$. (d) Tree structure

Figure 1. (a–c) Path between non-adjacent bases ($\mathcal{B} \rightarrow \mathcal{B}'$). $\mathcal{B}'$ can be generated from both $\mathcal{B}$ and $\mathcal{B}_{root}$, but it is *not* adjacent to $\mathcal{B}$ since $l(\mathcal{B}') = l(\mathcal{B})$. Note that Line 10 in Algorithm 1 cannot avoid this non-adjacent path since $\mathcal{V}(\mathcal{B}') \cup \{\mathbf{s}_2\} = \{\mathbf{s}_1, \mathbf{s}_2\} \neq \mathcal{V}(\mathcal{B}') = \{\mathbf{s}_1\}$. Panel (d) shows the relationship between the three bases during tree search. In the proposed Non-Adjacent Path Avoidance (NAPA) strategy, the path drawn in red is not followed. As we will show in Sec. 6, this simple idea provides a massive reduction in runtime of A* tree search.

the minimax problem (4) on $\mathcal{C}(\mathcal{B})\backslash\{\mathbf{s}\}$. In this way,

$$l(\mathcal{B}') = l(\mathcal{B}) + 1. \tag{9}$$

Iterating the $\mathbf{s}$ to be removed thus generates all the adjacent child bases of $\mathcal{B}$, which allows the tree to be explored.

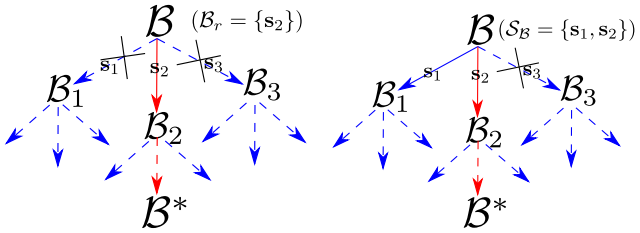
However, an important phenomenon that is ignored in Algorithm 1 is, while the above process generates all the adjacent child bases of $\mathcal{B}$, *not all $\mathcal{B}'$ generated in the process are adjacent child bases*. Figure 1 shows a concrete example from line fitting (2): from a root node $\mathcal{B}_{root}$, two child bases $\mathcal{B}$ and $\mathcal{B}'$ are generated by respectively removing points $\mathbf{s}_2$ and $\mathbf{s}_1$. However, by further removing $\mathbf{s}_1$ from $\mathcal{B}$ and solving (4) on $\mathcal{C}(\mathcal{B} \setminus \{\mathbf{s}_1\})$, we obtain $\mathcal{B}'$ again! Since $l(\mathcal{B}') = l(\mathcal{B})$, these two bases are not adjacent.

In general, non-adjacent paths occur in Algorithm 1 when some elements of $\mathcal{V}(\mathcal{B})$ are in $\mathcal{C}(\mathcal{B}')$ after solving the minimax problem on $\mathcal{C}(\mathcal{B} \setminus \{\mathbf{s}\})$. While inserting a non-adjacent $\mathcal{B}'$ into the queue does not affect global optimality, it does reduce efficiency. This is because the repeated basis check heuristic in Algorithm 1 assumes that the level of the child node $\mathcal{B}'$ is always lower than the parent $\mathcal{B}$ by 1; this assumption does not hold if the generated basis $\mathcal{B}'$ is not adjacent. More formally, if $\mathcal{B}'$ is not adjacent to $\mathcal{B}$, then

$$\mathcal{V}(\mathcal{B}) \cup \{\mathbf{s}\} \neq \mathcal{V}(\mathcal{B}') \tag{10}$$

and the repeated basis check in Line 8 in Algorithm 1 fails. Since the same $\mathcal{B}'$ could be generated from its "real" parent (e.g., in Figure 1, $\mathcal{B}'$ was also generated by $\mathcal{B}_{root}$), the same basis can be inserted into the queue more than once.

Since tree search only needs adjacent paths, we can safely skip traversing any non-adjacent path without affecting the final solution. To do this, we propose a Non-Adjacent Path Avoidance (NAPA) strategy for A* tree search; see Fig. 1(d). Given a basis $\mathcal{B}$, any non-adjacent basis generated from it cannot have a level that is higher than $l(\mathcal{B})$. Therefore, we can simply discard any newly generated basis $\mathcal{B}'$ (Line 12) if $l(\mathcal{B}') \leq l(\mathcal{B})$. Though one redundant minimax problem (4) still needs to be solved when finding $\mathcal{B}'$, a much larger cost for computing $e(\mathcal{B}')$ (which requires to solve multiple problems (4)) is saved along with all the computation required for traversing the children of $\mathcal{B}'$. The effectiveness of this strategy will be demonstrated later in Sec. 6.



(a) TOD (b) DIBP

Figure 2. (a) In TOD, on current node $\mathcal{B}$, if $\mathbf{s}_2$ is identified as the true outlier, then the shortest path towards a feasible basis $\mathcal{B}^*$ must pass through $\mathbf{s}_2$ (path rendered in red). All the the other $|\mathcal{B}| - 1$ branches (leading from $\mathbf{s}_1$ and $\mathbf{s}_3$ in this example) can be skipped. (b) In DIBP, instead of attempting to identify a single true outlier, a group $\mathcal{S}_\mathcal{B}$ that contains at least one true outlier ($\mathcal{S}_\mathcal{B} = \{\mathbf{s}_1, \mathbf{s}_2\}$ in this example) is identified; if this is successful, the other $|\mathcal{B}| - |\mathcal{S}_\mathcal{B}|$ paths (corresponding to $\mathbf{s}_3$ in this example) can be skipped. DIBP is more effective than TOD because it is easier to reject a subset than a single point as outlier; see Sec. 4.2 for details.

4. Dimension-insensitive branch pruning

Our second improvement to A* tree search lies in a new branch pruning technique. We first review the original method (TOD) and then describe our new technique.

4.1. Review of true outlier detection (TOD)

Referring to Line 8 in Algorithm 1 [6], let $\mathcal{F}^*$ be the largest feasible subset of $\mathcal{C}(\mathcal{B})$. A point $\mathbf{s} \in \mathcal{B}$ is said to be a true outlier if $\mathbf{s} \notin \mathcal{F}^*$, otherwise we call it a true inlier. Given an infeasible node $\mathcal{B}$, one of the elements in $\mathcal{B}$ must be a true outlier. The goal of TOD is to identify one such true outlier in $\mathcal{B}$. If $\mathbf{s} \in \mathcal{B}$ is successfully identified as a

true outlier, we can skip the child generation for all the other points in $\mathcal{B}$ without hurting optimality, since $\mathbf{s}$ must be on the shortest path to feasibility via $\mathcal{B}$; see Fig. 2(a). If such an $\mathbf{s}$ can be identified, the reduced subset $\mathcal{B}_r$ is simply $\{\mathbf{s}\}$.

The principle of TOD is as follows: define $h^*(\mathcal{B}|\mathbf{s})$ as the *minimum* number of data points that must be removed from $\mathcal{C}(\mathcal{B})$ to achieve feasibility, with $\mathbf{s}$ forced to be feasible. We can conclude that $\mathbf{s} \in \mathcal{B}$ is a true outlier if and only if

$$h^*(\mathcal{B}|\mathbf{s}) > h^*(\mathcal{B}); \tag{11}$$

see [6] for the formal proof. Intuitively, if $\mathbf{s}$ is a true inlier, forcing its feasibility will not change the value of h^*. On the other hand, if forcing $\mathbf{s}$ to be feasible leads to the above condition, $\mathbf{s}$ cannot be a true inlier.

Bound computation for TOD. Unsurprisingly $h^*(\mathcal{B}|\mathbf{s})$ is as difficult to compute as $h^*(\mathcal{B})$. To avoid directly computing $h^*(\mathcal{B}|\mathbf{s})$, TOD computes an admissible heuristic $h(\mathcal{B}|\mathbf{s})$ of $h^*(\mathcal{B}|\mathbf{s})$ and an upper bound $g(\mathcal{B})$ of $h^*(\mathcal{B})$. Given $\mathbf{s} \in \mathcal{B}$, $h(\mathcal{B}|\mathbf{s})$ and $g(\mathcal{B})$, if

$$h(\mathcal{B}|\mathbf{s}) > g(\mathcal{B}), \tag{12}$$

then it must hold that

$$h^*(\mathcal{B}|\mathbf{s}) \geq h(\mathcal{B}|\mathbf{s}) > g(\mathcal{B}) \geq h^*(\mathcal{B}), \tag{13}$$

which implies that $\mathbf{s}$ is a true outlier.

As shown in [6], $g(\mathcal{B})$ can be computed as a by-product of computing $h_{ins}(\mathcal{B})$, and $h(\mathcal{B}|\mathbf{s})$ is computed by a constrained version of h_{ins}, which we denote as $h_{ins}(\mathcal{B}|\mathbf{s})$. Computing $h_{ins}(\mathcal{B}|\mathbf{s})$ is done by the constrained version of Algorithm 2, where all minimax problems (4) required to solve are replaced by their constrained versions, which are in the following form:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \max_{\mathbf{s}_i \in \mathcal{S}^1} r(\boldsymbol{\theta}|\mathbf{s}_i), \tag{14a}$$

$$\text{s.t.} \quad r(\boldsymbol{\theta}|\mathbf{s}'_j) \leq \epsilon, \ \forall \mathbf{s}'_j \in \mathcal{S}'. \tag{14b}$$

The only difference between (14) and (4) is the constraint that all data in $\mathcal{S}'$ must be feasible. And similar to (4), (14) is also an LP-type problem which can be solved by standard solvers [10]. Similar as in (4) we also define $f(\mathcal{S}^1|\mathcal{S}')$ as the minimum objective value of (14) and $\boldsymbol{\theta}(\mathcal{S}^1|\mathcal{S}')$ as the corresponding optimal solution.

With the above definition, changing Algorithm 2 to its constrained version can be simply done by replacing $f(\mathcal{B})$ (Line 3) and $f(\mathcal{B}')$ (Line 10) by $f(\mathcal{B}|\{\mathbf{s}\})$ and $f(\mathcal{B}'|\{\mathbf{s}\})$.

Why is TOD ineffective? The effectiveness of TOD in accelerating Algorithm 1 depends on how frequent TOD can detect a true outlier. When a true outlier for $\mathcal{B}$ is detected, TOD prunes $|\mathcal{B}| - 1$ branches; on the flipside, if TOD cannot identify an $\mathbf{s} \in \mathcal{B}$ as the true outlier, the runtime to compute $h_{ins}(\mathcal{B}|\mathbf{s})$ will be wasted. In the worst case where no true outlier is identified for $\mathcal{B}$, Algorithm 2 has to be executed redundantly for $|\mathcal{B}|$ times. Whether TOD can find the true outlier is largely decided by how well $h_{ins}(\mathcal{B}|\mathbf{s})$ approximates $h^*(\mathcal{B}|\mathbf{s})$.

We now show that $h_{ins}(\mathcal{B}|\mathbf{s})$ is usually a poor estimator of $h^*(\mathcal{B}|\mathbf{s})$. Define $\mathcal{O}^*(\mathcal{B}|\mathbf{s})$ as the smallest subset that must be removed from $\mathcal{C}(\mathcal{B})\backslash\mathbf{s}$ to achieve feasibility, with $\mathbf{s}$ forced to be feasible, i.e., $|\mathcal{O}^*(\mathcal{B}|\mathbf{s})| = h^*(\mathcal{B}|\mathbf{s})$. Then, $h_{ins}(\mathcal{B}|\mathbf{s})$ and $h^*(\mathcal{B}|\mathbf{s})$ will be different if a basis $\mathcal{B}_{rem}$ removed during Algorithm 2 contains multiple elements in $\mathcal{O}^*(\mathcal{B}|\mathbf{s})$, since we only add 1 to h_{ins} when actually more than 1 points in $\mathcal{B}_{rem}$ should be removed. And the following lemma shows that the difference between $h_{ins}(\mathcal{B}|\mathbf{s})$ and $h^*(\mathcal{B}|\mathbf{s})$ will be too large for TOD to be effective if the rate of true outliers in $\mathcal{C}(\mathcal{B})$, i.e., $\frac{h^*(\mathcal{B})}{\mathcal{C}(\mathcal{B})}$, is too large.

Lemma 1. *Condition* (12) *is always false when*

$$\frac{h^*(\mathcal{B})}{\mathcal{C}(\mathcal{B})} \geq \frac{1}{\phi} \cdot \frac{|\mathcal{C}(\mathcal{B})| - 1}{|\mathcal{C}(\mathcal{B})|}, \tag{15}$$

where ϕ is the average *size of all $\mathcal{B}_{rem}$ during Algorithm 2.*

Proof. Since $h_{ins}(\mathcal{B}|\mathbf{s})$ is the number of $\mathcal{B}_{rem}$ during Algorithm 2, $h_{ins}(\mathcal{B}|\mathbf{s}) \cdot \phi \leq |\mathcal{C}(\mathcal{B})\backslash\{\mathbf{s}\}| = |\mathcal{C}(\mathcal{B})| - 1$. Hence,

$$h_{ins}(\mathcal{B}|\mathbf{s}) \leq \frac{|\mathcal{C}(\mathcal{B})| - 1}{\phi}, \tag{16}$$

Therefore, condition (12) can never be true if

$$h^*(\mathcal{B}) \geq \frac{|\mathcal{C}(\mathcal{B})| - 1}{\phi}. \tag{17}$$

Dividing both sides of (17) by $\mathcal{C}(\mathcal{B})$ leads to (15). □

Intuitively, when (15) happens, there are too many outliers in $\mathcal{C}(\mathcal{B})$ hence too many $\mathcal{B}_{rem}$ that include multiple elements in $\mathcal{O}^*(\mathcal{B}|\mathbf{s})$, making $h_{ins}(\mathcal{B}|\mathbf{s})$ too far from $h^*(\mathcal{B}|\mathbf{s})$.

In addition, ϕ is positively correlated with d, and in the worst case can be $d + 1$, which makes TOD sensitive to d. Figure 3 shows the effectiveness of TOD as a function of d, for problems with linear residual (2). As can be seen, the outlier rate where TOD can be effective reduces quickly with d ($< 15\%$ when $d \geq 7$). Note that since $g(\mathcal{B})$ is only an estimation of $h^*(\mathcal{B})$, the actual range where TOD is effective can be smaller than the region above the dashed line.

4.2. New pruning technique: DIBP

Due to the above limitation, TOD is often not effective in pruning; the cost to carry out Line 8 in Algorithm 1 is thus usually wasted. To address this issue, we propose a more effective branch pruning technique called DIBP (dimension-insensitive branch pruning).

DIBP extends the idea of TOD, where instead of searching for one true outlier, we search for a *subset* $\mathcal{S}_\mathcal{B}$ of $\mathcal{B}$ that

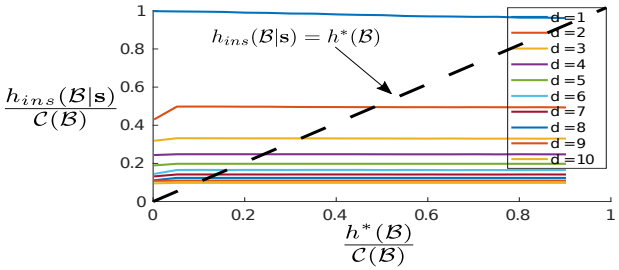


Figure 3. Effectiveness of TOD as a function of d. All problem instances are generated randomly and each solid curve contains data with true outlier rates $\frac{h^*(\mathcal{B})}{\mathcal{C}(\mathcal{B})}$ from 0 to 90%. Note that (15) is true for a d when the solid curve for the d is below the dashed line.

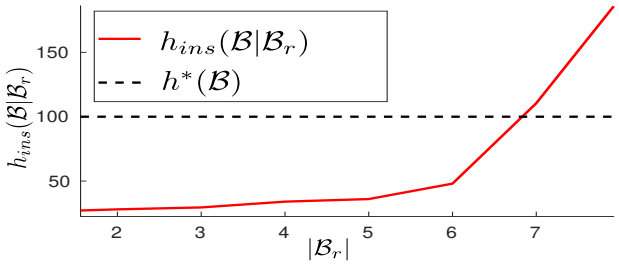


Figure 4. Effectiveness of DIBP when $d = 8$. $|\mathcal{C}(\mathcal{B})| = 200$. $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B})$ increases stably along with $|\mathcal{S}_\mathcal{B}|$ and is effective even when the true outlier rate is 90%. Though only the 50% case is shown, changing the outlier rate in practice merely affects the values of $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B})$ as long as the data distribution is similar.

must contain at least one true outlier. If such a subset can be identified, the children of $\mathcal{B}$ corresponding to removing points not in $\mathcal{S}_\mathcal{B}$ can be ignored during node expansion—again, this is because the shortest path to feasibility via $\mathcal{B}$ must go via $\mathcal{S}_\mathcal{B}$; Fig. 2(b) illustrates this idea.

To find such an $\mathcal{S}_\mathcal{B}$, we greedily add points from $\mathcal{B}$ into $\mathcal{S}_\mathcal{B}$ to see whether enforcing the feasibility of $\mathcal{S}_\mathcal{B}$ contradicts the following inequality

$$h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B}) > g(\mathcal{B}), \tag{18}$$

which is the extension of (12), with $h = h_{ins}$. Similar to $h_{ins}(\mathcal{B}|\mathbf{s})$, $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B})$ is computed by the constrained version of Algorithm 2 with $\mathcal{S}' = \mathcal{S}_\mathcal{B}$ in problem (14).

The insight is that by adding more and more constraints into problem (14), the average basis size ϕ will gradually reduce, making the right hand side of (15) increase until it exceeds the left hand side, so that even with large d, branch pruning will be effective with high true outlier rate. Figure 4 shows the effectiveness of DIBP for an 8-dimensional problem with linear residuals. Observe that $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B})$ increases steadily along with $|\mathcal{S}_\mathcal{B}|$ and can tolerate more than 90% of true outliers when $|\mathcal{S}_\mathcal{B}| = |\mathcal{B}| - 1 = 8$.

During DIBP, we want to add true outliers into $\mathcal{S}_\mathcal{B}$ as soon as possible, since (18) can never be true if $\mathcal{S}_\mathcal{B}$ contains no true outliers. To do so, we utilize the corresponding solution $\boldsymbol{\theta}_{g(\mathcal{B})}$ that leads to $g(\mathcal{B})$. During DIBP, the $\mathbf{s} \in \mathcal{B}$ with the largest residual $r(\boldsymbol{\theta}_{g(\mathcal{B})}|\mathbf{s})$ will be added into $\mathcal{S}_\mathcal{B}$ first, since a larger residual means a higher chance that $\mathbf{s}$ is a true outlier. In practice, this strategy often enables DIBP to find close to minimal-size $\mathcal{S}_\mathcal{B}$.

For problems with linear residuals, we can further compute an adaptive starting value $z(\mathcal{B})$ of $|\mathcal{S}_\mathcal{B}|$, where DIBP can safely skip the first $z(\mathcal{B})-1$ computations of $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B})$ without affecting the branch pruning result. The value of $z(\mathcal{B})$ should be $\max\{1, d+2-\frac{|\mathcal{C}(\mathcal{B})|-1}{g(\mathcal{B})}\}$. The reason is demonstrated in the following lemma:

Lemma 2. *For problems with linear residuals,* (18) *cannot be true unless*

$$|\mathcal{S}_\mathcal{B}| > d + 1 - \frac{|\mathcal{C}(\mathcal{B})| - 1}{g(\mathcal{B})}. \tag{19}$$

Proof. As in (16), we have $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B}) < \frac{|\mathcal{C}(\mathcal{B})|-1}{\phi}$. To ensure that (18) can be true, we must have $g(\mathcal{B}) < \frac{|\mathcal{C}(\mathcal{B})|-1}{\phi}$, which we rewrite as

$$\phi < \frac{|\mathcal{C}(\mathcal{B})| - 1}{g(\mathcal{B})}. \tag{20}$$

And for problems with linear residuals, (14) with $\mathcal{S}' = \mathcal{S}_\mathcal{B}$ is a linear program, whose optimal solution resides at a vertex of the feasible polytope [19, Chapter 13]. This means that for problem (14), the basis size plus the number of active constraints at the optimal solution must be $d + 1$. And since each absolute-valued constraint in (14b) can at most contribute one active linear constraint, the maximum number of active constraints is $|\mathcal{S}_\mathcal{B}|$. Thus during the computation of $h_{ins}(\mathcal{B}|\mathcal{S}_\mathcal{B})$, the average basis size $\phi \geq d + 1 - |\mathcal{S}_\mathcal{B}|$. Substituting this inequality into (20) results in (19). □

5. Main algorithm

Algorithm 3 summarizes the A* tree search algorithm with our new acceleration techniques. A reordering is done so that cheaper acceleration techniques are executed first. Specifically, given the current basis $\mathcal{B}$, we iterate through each element $\mathbf{s} \in \mathcal{B}$ and check first whether it leads to a repeated adjacent node and skip $\mathbf{s}$ if yes (Line 8). Otherwise, we check whether the node $\mathcal{B}'$ generated by $\mathbf{s}$ is non-adjacent to $\mathcal{B}$ and discard $\mathcal{B}'$ if yes (Line 11). If not, we insert $\mathcal{B}'$ into the queue since it cannot be pruned by other techniques. After that, we perform DIBP (Line 14) and skip the other elements in $\mathcal{B}$ if condition (18) is satisfied. Note that we can still add $\mathbf{s}$ into $\mathcal{S}_\mathcal{B}$ even though it leads to repeated bases. This strategy makes DIBP much more effective in practice.

6. Experiments

To demonstrate the effectiveness of our new techniques, we compared the following A* tree search variants:

Algorithm 3 A* tree search with NAPA and DIBP

Require: $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^N$, threshold ϵ.
Insert $\mathcal{B} = \tau(\mathcal{S})$ with priority $e(\mathcal{B})$ into queue q.
Initialize hash table T to NULL.
while q is not empty **do**
  Retrieve from q the $\mathcal{B}$ with the lowest $e(\mathcal{B})$.
  If $f(\mathcal{B}) \leq \epsilon$ **then** return $\mathcal{B}^* = \mathcal{B}$.
  $\mathcal{S}_\mathcal{B} \leftarrow \emptyset$; Sort $\mathcal{B}$ descendingly based on $r(\boldsymbol{\theta}_{g(\mathcal{B})}|\mathbf{s})$.
  for each $\mathbf{s} \in \mathcal{B}$ **do**
    if indices of $\mathcal{V}(\mathcal{B}) \cup \{\mathbf{s}\}$ do not exist in T **then**
      Hash indices of $\mathcal{V}(\mathcal{B}) \cup \{\mathbf{s}\}$ into T.
      $\mathcal{B}' \leftarrow \tau(\mathcal{C}(\mathcal{B}) \backslash \{\mathbf{s}\})$.
      if $l(\mathcal{B}') > l(\mathcal{B})$ **then**.
        $\mathcal{S}_\mathcal{B} \leftarrow \mathcal{S}_\mathcal{B} \cup \{\mathbf{s}\}$.
        Insert $\mathcal{B}'$ with priority $e(\mathcal{B}')$ into q.
        If $|\mathcal{S}_\mathcal{B}| = |\mathcal{B}| \vee$ (18) is true **then** break.
      end if
    else
      $\mathcal{S}_\mathcal{B} \leftarrow \mathcal{S}_\mathcal{B} \cup \{\mathbf{s}\}$.
    end if
  end for
end while
Return error (no inlier set of size greater than p).

- Original A* tree search (A*) [5].
- A* with TOD for branch pruning (A*-TOD) [6].
- A* with non-adjacenct path avoidance (A*-NAPA).
- A*-NAPA with TOD branch pruning (A*-NAPA-TOD).
- A*-NAPA with DIBP branch pruning (A*-NAPA-DIBP).

All variants were implemented in MATLAB 2018b, based on the original code of A*. For problems with linear residuals, we use the self-implemented vertex-to-vertex algorithm [3] to solve the minimax problems (4) and (14). And in the non-linear case, these two problems were solved by the matlab function `fminimax`. All experiments were executed on a laptop with Intel Core 2.60GHz i7 CPU, 16GB RAM and Ubuntu 14.04 OS.

6.1. Controlled experiment on synthetic data

To analyze the effect of o and N to different methods, we conducted a controlled experiment on the 8-dimensional robust linear regression problem with different N and o. The residual of linear regression is in the form of (2). To generate data $\mathcal{S} = \{\mathbf{a}_i, b_i\}_{i=1}^N$, a random model $\boldsymbol{\theta} \in \mathbb{R}^d$ was first generated and N data points that perfectly fit the model were randomly sampled. Then, we randomly picked $N - o$ points as inliers and assigned to the b_i of these points noise uniformly distributed between $[-0.1, 0.1]$. Then we assigned to the other o points noise uniformly distributed from $[-5, -0.1) \cup (0.1, 5]$ to create a controlled number of outliers. The inlier threshold ϵ was set to 0.1.

To verify the superior efficiency of tree search compared to other types of globally optimal methods, we also tested the Mixed Integer Programming-based BnB algorithm (MIP) [28] in this experiment. The state-of-the-art Gurobi solver was used as the optimizer for MIP. MIP was parallelized by Gurobi using 8 threads, while all tree search methods were executed sequentially.

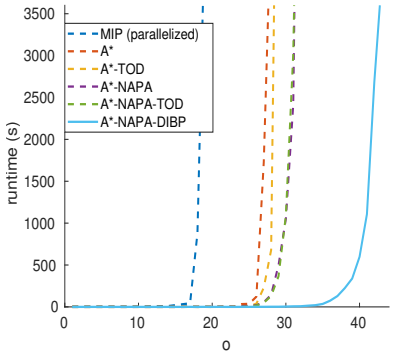
As shown in Figure 5, all A* tree search variants are much faster than MIP, even though MIP was significantly accelerated by parallel computing. Both NAPA and DIBP brought considerable acceleration to A* tree search, which can be verified by the gaps between the variants with and without these techniques. Note that when $N = 200$, A*-NAPA had similar performance with and without TOD, while DIBP provided stable and significant acceleration for all data.

Interestingly, having a larger N made A* tree search efficient for a much larger o. This can be explained by condition (15). With the same o, a larger N meant a lower true outlier rate, which made (15) less likely.

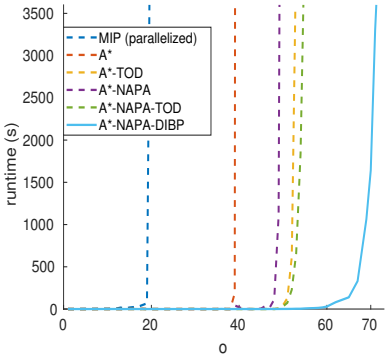
6.2. Linearized fundamental matrix estimation

Experiments were also conducted on real data. We executed all tree seach variants for linearized fundamental matrix estimation [6], which used the algebaric error [13, Sec.11.3] as the residual and ignored the non-convex rank-2 constraints. 5 image pairs (the first 5 crossroads) were selected from the sequence `00` of the KITTI Odometry dataset [12]. For each image pair, the input was a set of SIFT [17] feature matches generated using VLFeat [27]. The inlier threshold ϵ was set to 0.03 for all image pairs.

The result is shown in Table 1. We also showed the number of unique nodes (NUN) generated and the number of branch pruning steps (NOBP) executed before the termination of each algorithm. A*-NAPA-DIBP found the optimal solution in less than 10s for all data, while A* and A*-TOD often failed to finish in 2 hours. A*-NAPA-DIBP was faster by more than 500 times on all data compared to the fastest method among A* and A*-TOD. For the effectiveness of each technique, applying NAPA to A* often resulted in more than 10x acceleration. And applying DIBP further sped up A*-NAPA by more than 1000x on challenging data (e.g. `Frame-198-201`). This significant acceleration is because many elements in $\mathcal{S}_\mathcal{B}$ were the ones that led to redundant nodes, which made most non-redundant paths effectively pruned. TOD was much less effective than DIBP and introduced extra runtime to A*-NAPA on `Frame-104-108` and `Frame-198-201`. We also attached o_{LRS}, the *estimated* number of outliers returned from LO-RANSAC [8], which is an effective RANSAC variant. None of the LO-RANSAC results were optimal. A visualization of the tree search result is shown in Figure 6.



(a) $N = 200$



(b) $N = 400$

Figure 5. Runtime vs o for robust linear regression on synthetic data. $d = 8$.

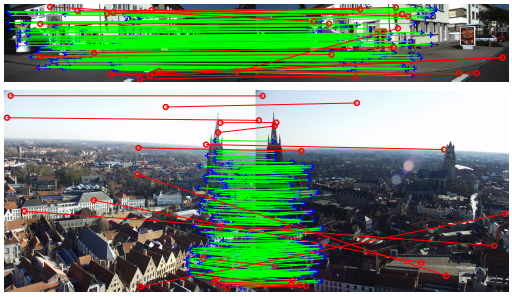


Figure 6. (Top) Fundamental matrix estimation result of A*-NAPA-DIBP on `Frame-738-742`. (Bottom) Homography estimation result of A*-NAPA-DIBP on data `BruggeTower`. The inliers (in green) in the top figure were down-sampled to 100 for clarity.

data $d = 8$	`Frame-104-108` $o = 13$ ($o_{LRS} = 23$); $N = 302$		`Frame-198-201` $o = 13$ ($o_{LRS} = 19$); $N = 309$		`Frame-417-420` $o = 19$ ($o_{LRS} = 23$); $N = 385$		`Frame-579-582` $o = 22$ ($o_{LRS} = 25$); $N = 545$		`Frame-738-742` $o = 14$ ($o_{LRS} = 32$); $N = 476$	
	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)
A*	163232/0	> 6400	169369/0	> 6400	144560/0	> 6400	136627/0	> 6400	160756/0	> 6400
A*-TOD	134589/119871	> 6400	129680/126911	> 6400	80719/92627	3712.99	55764/58314	2709.21	49586/50118	1729.34
A*-NAPA	35359/0	561.81	23775/0	351.07	175806/0	5993.68	147200/0	> 6400	29574/0	471.15
A*-NAPA-TOD	33165/22275	770.08	19308/13459	451.39	15310/10946	429.06	15792/12073	576.82	14496/10752	373.36
A*-NAPA-DIBP	205/311	**7.63**	105/160	**3.88**	172/216	**6.85**	60/84	**3.49**	52/77	**2.00**
A*-NAPA-DIBP vs previous best method	best previous method A*/A*-TOD	faster by $> \mathbf{839x}$	best previous method A*/A*-TOD	faster by $> \mathbf{1648x}$	best previous method A*-TOD	faster by **541x**	best previous method A*-TOD	faster by **775x**	best previous method A*-TOD	faster by **864x**

Table 1. Linearized fundamental matrix estimation result. The names of the data are the image indices in the sequence. o_{LRS} is the *estimated* outlier number returned by LO-RANSAC. NUN: number of unique nodes generated. NOBP: number of branch pruning steps executed. The last row shows how much faster A*-NAPA-DIBP was, compared to the fastest previously proposed variants (A* and A*-TOD).

data $d = 8$	`Adam` $o = 38$ ($o_{LRS} = 40$); $N = 282$		`City` $o = 19$ ($o_{LRS} = 22$); $N = 87$		`Boston` $o = 43$ ($o_{LRS} = 44$); $N = 678$		`Brussels` $o = 9$ ($o_{LRS} = 25$); $N = 231$		`BruggeTower` $o = 17$ ($o_{LRS} = 26$); $N = 208$	
	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)	NUN/NOBP	runtime (s)
A*	224/0	538.91	7072/0	> 6400	406/0	2455.03	397/0	437.25	5003/0	> 6400
A*-TOD	38/37	**156.98**	462/514	910.51	7/6	**74.63**	359/281	499.77	333/260	298.39
A*-NAPA	168/0	404.77	6481/0	> 6400	234/0	1284.14	264/0	268.85	3731/0	4740.68
A*-NAPA-TOD	38/37	**156.98**	286/241	485.36	7/6	**74.63**	249/191	297.91	201/151	161.95
A*-NAPA-DIBP	38/37	**156.98**	34/40	**64.44**	7/6	**74.63**	30/42	**50.13**	40/48	**68.20**
A*-NAPA-DIBP vs previous best method	best previous method A*-TOD	faster by same runtime	best previous method A*-TOD	faster by **13.1x**	best previous method A*-TOD	faster by same runtime	best previous method A*	faster by **7.7x**	best previous method A*-TOD	faster by **3.4x**

Table 2. Homography estimation result. o_{LRS} is the *estimated* outlier number returned by LO-RANSAC. NUN: number of unique nodes generated. NOBP: number of branch pruning steps executed. The last row shows how much faster A*-NAPA-DIBP was, compared to the fastest previously proposed variants (A* and A*-TOD).

6.3. Homography estimation (non-linear)

To test all methods on non-linear problems, another experiment for homography estimation [13] was done on "homogr" dataset[1]. As before, we picked 5 image pairs, computed the SIFT matches and used them as the input data. The transfer error in one image [13] was used as the residual, which was in the form of (3). ϵ was set to 4 pixels.

Table 2 shows the result of all methods. Compared to the linear case, solving non-linear minimax problems (4) and (14) was much more time-consuming (can be 100x slower with `fminimax`). Thus with similar NUN and NOBP, the runtime was much larger. However, the value of ϕ in the non-linear case was usually also much smaller, which made the heuristic h_{ins} and in turn all branch pruning techniques much more effective than in the linear case. And for easy data such as `Boston` and `Adam`, performing either TOD or DIBP was enough to achieve the highest speed. Nonetheless, DIBP was still much more effective than TOD on other data. And DIBP never slowed down the A* tree search as TOD sometimes did (e.g., in `Brussels`). A*-NAPA-DIBP remained fastest on all image pairs. An example of the visual result is provided in Figure 6.

[1] `http://cmp.felk.cvut.cz/data/geometry2view/index.xhtml`

7. Conclusion

We presented two new acceleration techniques for consensus maximization tree search. The first avoids redundant non-adjacent paths that exist in the consensus maximization tree structure. The second makes branch pruning much less sensitive to the problem dimension, and therefore much more reliable. The significant acceleration brought by the two techniques contributes a solid step towards practical and globally optimal consensus maximization.

Acknowledgements. We thank Dr. Nan Li for his valuable suggestions.

References

[1] Nina Amenta, Marshall Bern, and David Eppstein. Optimal point placement for mesh smoothing. *Journal of Algorithms*, 30(2):302–322, 1999.

[2] Zhipeng Cai, Tat-Jun Chin, Huu Le, and David Suter. Deterministic consensus maximization with biconvex programming. In *European Conference on Computer Vision (ECCV)*, 2018.

[3] E. W. Cheney. *Introduction to Approximation Theory*. McGraw-Hill, 1966.

[4] Tat-Jun Chin, Zhipeng Cai, and Frank Neumann. Robust fitting in computer vision: Easy or hard? In *European Conference on Computer Vision (ECCV)*, 2018.

[5] Tat-Jun Chin, Pulak Purkait, Anders Eriksson, and David Suter. Efficient globally optimal consensus maximisation with tree search. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[6] Tat-Jun Chin, Pulak Purkait, Anders Eriksson, and David Suter. Efficient globally optimal consensus maximisation with tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(4):758–772, 2017.

[7] Ondrej Chum and Jiri Matas. Matching with prosac-progressive sample consensus. In *Computer Vision and Pattern Recognition (CVPR)*, 2005.

[8] Ondřej Chum, Jiří Matas, and Josef Kittler. Locally optimized RANSAC. In *Joint Pattern Recognition Symposium*, 2003.

[9] Olof Enqvist, Erik Ask, Fredrik Kahl, and Kalle Åström. Robust fitting for multiple view geometry. In *European Conference on Computer Vision (ECCV)*, 2012.

[10] David Eppstein. Quasiconvex programming. *Combinatorial and Computational Geometry*, 52(3):287–331, 2005.

[11] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[13] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[14] Huu Le, Tat-Jun Chin, and David Suter. An exact penalty method for locally convergent maximum consensus. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[15] Hongdong Li. A practical algorithm for L_∞ triangulation with outliers. In *Computer Vision and Pattern Recognition (CVPR)*, 2007.

[16] Hongdong Li. Consensus set maximization with guaranteed global optimality for robust geometry estimation. In *International Conference on Computer Vision (ICCV)*, 2009.

[17] David G Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV)*, 1999.

[18] Jiří. Matoušek. On geometric optimization with few violated constraints. *Discrete and Computational Geometry*, 14(4):365–384, 1995.

[19] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[20] Carl Olsson, Olof Enqvist, and Fredrik Kahl. A polynomial-time bound for matching and registration with outliers. In *Computer Vision and Pattern Recognition (CVPR)*, 2008.

[21] Carl Olsson, Anders P Eriksson, and Fredrik Kahl. Efficient optimization for L_∞-problems using pseudoconvexity. In *International Conference on Computer Vision (ICCV)*, 2007.

[22] Alvaro Parra Bustos and Tat-Jun Chin. Guaranteed outlier removal for rotation search. In *International Conference on Computer Vision (ICCV)*, 2015.

[23] Pulak Purkait, Christopher Zach, and Anders Eriksson. Maximum consensus parameter estimation by reweighted L1 methods. In *Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, 2017.

[24] Rahul Raguram, Ondrej Chum, Marc Pollefeys, Jiri Matas, and Jan-Michael Frahm. USAC: a universal framework for random sample consensus. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(8):2022–2038, 2013.

[25] Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *Annual Symposium on Theoretical Aspects of Computer Science*, 1992.

[26] Ben J Tordoff and David W Murray. Guided-MLESAC: Faster image transform estimation by using matching priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(10):1523–1535, 2005.

[27] Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms. In *ACM International Conference on Multimedia*, 2010.

[28] Yinqiang Zheng, Shigeki Sugimoto, and Masatoshi Okutomi. Deterministically maximizing feasible subsystems for robust model fitting with unit norm constraints. In *Computer Vision and Pattern Recognition (CVPR)*, 2011.