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](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 \( S = \{s_i\}_{i=1}^N \), we seek the model \( \theta \in \mathbb{R}^d \) that is consistent with the largest subset of the data. Formally, we solve

\[
\max_{\theta} c(\theta|S) = \sum_{i=1}^N I(r(\theta|s_i) \leq \epsilon), \quad (1)
\]

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

Consensus maximization is NP-hard [4], hence, suboptimal 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 only on 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 \( d \) (\( \geq 6 \)) and \( o \) (\( \geq 10 \)).

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 \( \theta \). 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 \( r(\theta|s_i) \) to be pseudo-convex [6]. A simple example is the linear regression residual

\[
r(\theta|s_i) = |a_i^T \theta - b_i|, \tag{2}
\]

where each datum \( s_i = \{a_i, b_i\} \), \( a_i \in \mathbb{R}^d \) and \( b_i \in \mathbb{R} \). Another example is the residual used in common multiview geometry problems [21, 2], which are of the form

\[
r(\theta|s_i) = \frac{\|A_i^T \theta - b_i\|_p}{c_i^T \theta - d_i}, \tag{3}
\]

where each datum \( s_i = \{A_i, b_i, c_i, d_i\} \), \( A_i \in \mathbb{R}^{d \times m} \), \( b_i \in \mathbb{R}^m \), \( c_i \in \mathbb{R}^d \) and \( d_i \in \mathbb{R} \). Usually, \( p \) is 1, 2 or \( \infty \).

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

\[
\text{minimize } \max_{\tau \in S^1} r(\theta|s_i). \tag{4}
\]

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

Throughout the paper, we will assume that \( r(\cdot) \) is pseudo-convex and \( S \) is non-degenerate (otherwise infinitesimal 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, 110], 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 \( S^1 \subseteq S^2 \subseteq S \), \( f(S^1) \leq f(S^2) \leq f(S) \).

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

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

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

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 \( B \) is defined as \( V(B) = \{s_i \in S | r(\theta(B)|s_i) > f(B)\} \). We call \( l(B) = |V(B)| \) the level of \( B \) and \( C(B) = S \setminus V(B) \) the coverage of \( B \).

By the above definition,

\[
c(\theta(B)|S) = |S| - l(B). \tag{5}
\]

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

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

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

\[
B^* = \arg \min_{B \subseteq S} l(B), \text{ s.t. } f(B) \leq \epsilon, \tag{6}
\]

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

2.3. A* tree search algorithm

Matoušek [13] showed that the set of bases for an LP-type problem can be arranged in a tree, where the root node is \( \tau(S) \), and the level occupied by a node \( B \) on the tree is \( l(B) = |V(B)| \). Another key insight is that there exists a path from \( \tau(S) \) 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: \( S = \{s_i\}_{i=1}^{N} \), threshold \( \epsilon \).
1. Insert \( B = \tau(S) \) with priority \( e(B) \) into queue \( q \).
2. Initialize hash table \( T \) to NULL.
3. while \( q \) is not empty do
4. Retrieve from \( q \) the \( B \) with the lowest \( e(B) \).
5. if \( f(B) \leq \epsilon \) then
6. return \( B^* = B \).
7. end if
8. \( B_r \leftarrow \text{Attempt to reduce } B \text{ by TOD method.} \)
9. for each \( s \in B_r \) do
10. if indices of \( V(B) \cup \{s\} \) do not exist in \( T \) then
11. Hash indices of \( V(B) \cup \{s\} \) into \( T \).
12. \( B' \leftarrow \tau(C(B) \setminus \{s\}) \).
13. Insert \( B' \) with priority \( e(B') \) 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 \( B' \) and \( B \) are adjacent if \( V(B') = V(B) \cup \{s\} \) for some \( s \in B \).

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

The evaluation value is defined as

\[
e(B) = l(B) + h(B),
\]

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

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

Note that setting \( e(B) = l(B) \) (i.e., \( h(B) = 0 \)) for all \( B \) reduces A* search to breadth-first search (BFS). With an admissible heuristic, A* search is guaranteed to always find \( B^* \) 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 \( F \subseteq C(B) \) is found. After that, the algorithm iteratively inserts each removed basis point \( s \) back into \( F \). If the insertion of \( s \) makes \( F \) infeasible, \( \tau(F \cup \{s\}) \) is removed from the expanded \( F \) and the heuristic value \( h_{ins} \) is increased by 1.

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

2.4. Avoiding redundant node expansions

Algorithm 1 employs two strategies to avoid redundant node expansions. In Line 8 before expanding \( B \), a fast heuristic called True Outlier Detection (TOD) [6] is used to attempt to identify and remove true outliers from \( B \) (more details in Sec. 4), which has the potential to reduce the size of the branch starting from \( B \). 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. 5).

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 \( B \) and \( B' \) to be adjacent, their violation sets \( V(B) \) and \( V(B') \) must differ by one point; in other words, it must hold that

\[
|l(B') - l(B)| = 1.
\]

Given a \( B \), Line 12 in Algorithm 1 generates an adjacent “child” basis of \( B \) by removing a point \( s \) from \( B \) and solving
Avoidance (NAPA) strategy for A* tree search; see Fig. 1(d).

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 $B$, any non-adjacent basis generated from it cannot have a level that is higher than $l(B)$. Therefore, we can simply discard any newly generated basis $B'$ (Line 12) if $l(B') \leq l(B)$. Though one redundant minimax problem (Fig. 1(b)) still needs to be solved when finding $B'$, a much larger cost for computing $c(B')$ (which requires to solve multiple problems (Fig. 1(b))) is saved along with all the computation required for traversing the children of $B'$. The effectiveness of this strategy will be demonstrated later in Sec. 6.

**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, let $F^*$ be the largest feasible subset of $C(B)$. A point $s \in B$ is said to be a true outlier if $s \notin F^*$, otherwise we call it a true inlier.

Given an infeasible node $B$, one of the elements in $B$ must be a true outlier. The goal of TOD is to identify one such true outlier in $B$. If $s \in B$ 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 $s$ must be on the shortest path to feasibility via $\mathcal{B}$; see Fig. 2(a). If such an $s$ can be identified, the reduced subset $\mathcal{B}_r$ is simply $\{s\}$.

The principle of TOD is as follows: define $h^*(\mathcal{B}|s)$ as the minimum number of data points that must be removed from $\mathcal{C}(\mathcal{B})$ to achieve feasibility, with $s$ forced to be feasible. We can conclude that $s \in \mathcal{B}$ is a true outlier if and only if
\[
h^*(\mathcal{B}|s) > h^*(\mathcal{B});
\] (11)
see [6] for the formal proof. Intuitively, if $s$ is a true outlier, forcing its feasibility will not change the value of $h^*$. On the other hand, if forcing $s$ to be feasible leads to the above condition, $s$ cannot be a true outlier.

**Bound computation for TOD.** Unsurprisingly $h^*(\mathcal{B}|s)$ is as difficult to compute as $h^*(\mathcal{B})$. To avoid directly computing $h^*(\mathcal{B}|s)$, TOD computes an admissible heuristic $h(\mathcal{B}|s)$ of $h^*(\mathcal{B}|s)$ and an upper bound $g(\mathcal{B})$ of $h^*(\mathcal{B})$. Given $s \in \mathcal{B}$, $h(\mathcal{B}|s)$ and $g(\mathcal{B})$, if
\[
h(\mathcal{B}|s) > g(\mathcal{B}),
\] (12)
then it must hold that
\[
h^*(\mathcal{B}|s) \geq h(\mathcal{B}|s) > g(\mathcal{B}) \geq h^*(\mathcal{B}),
\] (13)
which implies that $s$ 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 $h(\mathcal{B}|s)$ is computed by a constrained version of $h_{ins}$, which we denote as $h_{ins}^*(\mathcal{B}|s)$. Computing $h_{ins}(\mathcal{B}|s)$ 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:

\[
\begin{align*}
\text{minimize} & \quad \max_{s_i \in \mathcal{S}_i} r(\theta|s_i), \\
\text{s.t.} & \quad r(\theta|s_{i'}) \leq \epsilon, \forall s_{i'} \in \mathcal{S}'.
\end{align*}
\] (14a, 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 $f(\mathcal{S}_i|\mathcal{S}')$ as the minimum objective value of (14) and $\theta(\mathcal{S}_i|\mathcal{S}')$ 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 $f(\mathcal{B} \setminus \{s\})$ and $f(\mathcal{B}' \setminus \{s\})$.

**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 $s \in \mathcal{B}$ as the true outlier, the runtime to compute $h_{ins}(\mathcal{B}|s)$ 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 $h_{ins}(\mathcal{B}|s)$ approximates $h^*(\mathcal{B}|s)$.

We now show that $h_{ins}(\mathcal{B}|s)$ is usually a poor estimator of $h^*(\mathcal{B}|s)$. Define $O^*(\mathcal{B}|s)$ as the smallest subset that must be removed from $\mathcal{C}(\mathcal{B}) \setminus \{s\}$ to achieve feasibility, with $s$ forced to be feasible, i.e., $|O^*(\mathcal{B}|s)| = h^*(\mathcal{B}|s)$. Then, $h_{ins}(\mathcal{B}|s)$ and $h^*(\mathcal{B}|s)$ will be different if a basis $\mathcal{B}_{rem}$ removed during Algorithm 2 contains multiple elements in $O^*(\mathcal{B}|s)$, 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 $h_{ins}(\mathcal{B}|s)$ and $h^*(\mathcal{B}|s)$ will be too large for TOD to be effective if the rate of true outliers in $\mathcal{C}(\mathcal{B})$, i.e., $\frac{h^*(\mathcal{B}|s)}{|\mathcal{C}(\mathcal{B})|}$, is too large.

**Lemma 1.** Condition (12) is always false when
\[
\frac{h^*(\mathcal{B}|s)}{|\mathcal{C}(\mathcal{B})|} \geq \frac{1}{\phi} \cdot \frac{|\mathcal{C}(\mathcal{B})| - 1}{|\mathcal{C}(\mathcal{B})|},
\] (15)
where $\phi$ is the average size of all $\mathcal{B}_{rem}$ during Algorithm 2.

**Proof.** Since $h_{ins}(\mathcal{B}|s)$ is the number of $\mathcal{B}_{rem}$ during Algorithm 2
\[
h_{ins}(\mathcal{B}|s) \cdot \phi \leq |\mathcal{C}(\mathcal{B}) \setminus \{s\}| = |\mathcal{C}(\mathcal{B})| - 1.
\]
Hence,
\[
h_{ins}(\mathcal{B}|s) \leq \frac{|\mathcal{C}(\mathcal{B})| - 1}{\phi},
\] (16)
Therefore, condition (12) can never be true if
\[
h^*(\mathcal{B}|s) \geq \frac{|\mathcal{C}(\mathcal{B})| - 1}{\phi}.
\] (17)
Dividing both sides of (17) by $|\mathcal{C}(\mathcal{B})|$ leads to (15). \qed

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 $O^*(\mathcal{B}|s)$, making $h_{ins}(\mathcal{B}|s)$ too far from $h^*(\mathcal{B}|s)$.

In addition, $\phi$ 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 $d > 7$). 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 $\mathcal{S}_B$ of $\mathcal{B}$
must contain at least one true outlier. If such a subset can be identified, the children of $B$ corresponding to removing points not in $S_B$ can be ignored during node expansion—again, this is because the shortest path to feasibility via $B$ must go via $S_B$; Fig. 2(b) illustrates this idea.

To find such an $S_B$, we greedily add points from $B$ into $S_B$ to see whether enforcing the feasibility of $S_B$ contradicts the following inequality

$$h_{ins}(B|S_B) > g(B), \quad (18)$$

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

The insight is that by adding more and more constraints into problem (14), the average basis size $\phi$ 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 $h_{ins}(B|S_B)$ increases steadily along with $|S_B|$ and can tolerate more than 90% of true outliers when $|S_B| = |B| - 1 = 8$.

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

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

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

$$|S_B| > d + 1 - \frac{|C(B)| - 1}{g(B)}. \quad (19)$$

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

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

And for problems with linear residuals, (14) with $S' = S_B$ 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 $|S_B|$. Thus during the computation of $h_{ins}(B|S_B)$, the average basis size $\phi \geq d + 1 - |S_B|$. Substituting this inequality into (20) results in (19). \qed

**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 $B$, we iterate through each element $s \in B$ and check first whether it leads to a repeated adjacent node and skip $s$ if yes (Line 5). Otherwise, we check whether the node $B'$ generated by $s$ is non-adjacent to $B$ and discard $B'$ if yes (Line 11). If not, we insert $B'$ into the queue since it cannot be pruned by other techniques. After that, we perform DIBP (Line 13) and skip the other elements in $B$ if condition (18) is satisfied. Note that we can still add $s$ into $S_B$ 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:
was first generated and we conducted a controlled experiment on the 8-dimensional
Algorithm 3 A* tree search with NAPA and DIBP
Require: $S = \{s_i\}_{i=1}^N$, threshold $\epsilon$.
1. Insert $B = \tau(S)$ with priority $\epsilon(B)$ into queue $q$.
2. Initialize hash table $T$ to NULL.
3. while $q$ is not empty do
4. Retrieve from $q$ the $B$ with the lowest $\epsilon(B)$.
5. if $f(B) \leq \epsilon$ then return $B^* = B$.
6. $S_B \leftarrow \emptyset$; Sort $B$ descendingly based on $\tau(\theta_B(B)|s)$.
7. for each $s \in B$ do
8. if indices of $V(B) \cup \{s\}$ do not exist in $T$ then
9. Hash indices of $V(B) \cup \{s\}$ into $T$.
10. $B' \leftarrow \tau(C(B)|\{s\})$.
11. if $l(B') > l(B)$ then.
12. $S_B \leftarrow S_B \cup \{s\}$.
13. Insert $B'$ with priority $\epsilon(B')$ into $q$.
14. if $|S_B| = |B| \lor (18)$ is true then break.
15. end if
16. else.
17. $S_B \leftarrow S_B \cup \{s\}$.
18. end if
19. end for
20. end while
21. 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-adjacent 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 $S = \{a_i, b_i\}_{i=1}^N$, a random model $\theta \in \mathbb{R}^d$ 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 $\epsilon$ 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 search variants for linearized fundamental matrix estimation [6], which used the algebraic 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 $\epsilon$ 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 than MIP 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 $T_B$ 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. 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 $T_B$ 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. 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 $T_B$ 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. 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 $T_B$ 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. 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 $T_B$ 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.
Figure 5. Runtime vs $\alpha$ 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 Brusseltower. The inliers (in green) in the top figure were down-sampled to 100 for clarity.

Table 1. Linearized fundamental matrix estimation result. The names of the data are the image indices in the sequence. $\text{OL}_\text{RS}$ 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 $A^*-TOD$).

<table>
<thead>
<tr>
<th>$\text{data}$</th>
<th>$\text{Frame-104-108}$</th>
<th>$\text{Frame-109-211}$</th>
<th>$\text{Frame-411-425}$</th>
<th>$\text{Frame-579-582}$</th>
<th>$\text{Frame-738-742}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 8$</td>
<td>$\alpha = 13$ ($q_{xy} = 23$); $N = 302$</td>
<td>$\alpha = 13$ ($q_{xy} = 19$); $N = 308$</td>
<td>$\alpha = 19$ ($q_{xy} = 23$); $N = 385$</td>
<td>$\alpha = 22$ ($q_{xy} = 25$); $N = 545$</td>
<td>$\alpha = 14$ ($q_{xy} = 32$); $N = 476$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>734320</td>
<td>716869</td>
<td>404518</td>
<td>364392</td>
<td>212340</td>
</tr>
<tr>
<td>$A^*-TOD$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
</tr>
<tr>
<td>$A^*-NAPA$</td>
<td>134589/110871</td>
<td>129860/126911</td>
<td>135100/10646</td>
<td>137030/10173</td>
<td>138270/9100</td>
</tr>
<tr>
<td>$A^*-NAPA$-DIBP</td>
<td>553990</td>
<td>56081</td>
<td>19030/30549</td>
<td>451.9</td>
<td>576.8</td>
</tr>
<tr>
<td>$A^<em>-NAPA$-DIBP vs $A^</em>$ best previous method faster by $839$</td>
<td>$A^*-TOD$ best previous method faster by $10448$</td>
<td>$A^*-TOD$ best previous method faster by $544$</td>
<td>$A^*-TOD$ best previous method faster by $779$</td>
<td>$A^*-TOD$ best previous method faster by $864$</td>
<td></td>
</tr>
<tr>
<td>$A^<em>-NAPA$-DIBP vs $A^</em>-TOD$ best previous method faster by $&gt; 839$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $&gt; 10448$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $544$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $779$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $864$</td>
<td></td>
</tr>
<tr>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>165232</td>
<td>106530</td>
<td>134500</td>
<td>130627/98</td>
<td>136590</td>
</tr>
<tr>
<td>$A^*-TOD$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
<td>$&gt; 6400$</td>
</tr>
<tr>
<td>$A^*-NAPA$</td>
<td>134589/110871</td>
<td>129860/126911</td>
<td>135100/10646</td>
<td>137030/10173</td>
<td>138270/9100</td>
</tr>
<tr>
<td>$A^*-NAPA$-DIBP</td>
<td>553990</td>
<td>56081</td>
<td>19030/30549</td>
<td>451.9</td>
<td>576.8</td>
</tr>
<tr>
<td>$A^<em>-NAPA$-DIBP vs $A^</em>$ best previous method faster by $839$</td>
<td>$A^*-TOD$ best previous method faster by $10448$</td>
<td>$A^*-TOD$ best previous method faster by $544$</td>
<td>$A^*-TOD$ best previous method faster by $779$</td>
<td>$A^*-TOD$ best previous method faster by $864$</td>
<td></td>
</tr>
<tr>
<td>$A^<em>-NAPA$-DIBP vs $A^</em>-TOD$ best previous method faster by $&gt; 839$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $&gt; 10448$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $544$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $779$</td>
<td>$A^*-NAPA$-DIBP best previous method faster by $864$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Homography estimation result. $\text{OL}_\text{RS}$ 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 $A^*-TOD$).

<table>
<thead>
<tr>
<th>$\text{data}$</th>
<th>$\text{Adam}$</th>
<th>$\text{City}$</th>
<th>$\text{Boston}$</th>
<th>$\text{Brussels}$</th>
<th>$\text{Brusseltower}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = 8$</td>
<td>$\alpha = 38$ ($q_{xy} = 40$); $N = 282$</td>
<td>$\alpha = 38$ ($q_{xy} = 22$); $N = 87$</td>
<td>$\alpha = 43$ ($q_{xy} = 44$); $N = 678$</td>
<td>$\alpha = 9$ ($q_{xy} = 25$); $N = 231$</td>
<td>$\alpha = 17$ ($q_{xy} = 32$); $N = 208$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>2240</td>
<td>534.91</td>
<td>707290</td>
<td>40660</td>
<td>2455.05</td>
</tr>
<tr>
<td>$A^*-TOD$</td>
<td>36/37</td>
<td>156.98</td>
<td>462514</td>
<td>76</td>
<td>74.63</td>
</tr>
<tr>
<td>$A^*-NAPA$</td>
<td>16/80</td>
<td>484.77</td>
<td>6481.00</td>
<td>2340</td>
<td>1284.14</td>
</tr>
<tr>
<td>$A^*-NAPA$-DIBP</td>
<td>38/37</td>
<td>156.98</td>
<td>286241</td>
<td>76</td>
<td>74.63</td>
</tr>
<tr>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>36/37</td>
<td>156.98</td>
<td>340.00</td>
<td>76</td>
<td>74.63</td>
</tr>
<tr>
<td>$A^*-NAPA$</td>
<td>16/80</td>
<td>484.77</td>
<td>6481.00</td>
<td>2340</td>
<td>1284.14</td>
</tr>
<tr>
<td>$A^*-NAPA$-DIBP</td>
<td>38/37</td>
<td>156.98</td>
<td>286241</td>
<td>76</td>
<td>74.63</td>
</tr>
<tr>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
<td>$\text{runtime (s)}$</td>
<td>$\text{NUN/NOBP}$</td>
</tr>
<tr>
<td>$A^<em>-NAPA$-DIBP vs $A^</em>$ previous best method faster by $13.1x$</td>
<td>$A^*-TOD$ previous best method faster by $7.7x$</td>
<td>$A^*-TOD$ previous best method faster by $3.4x$</td>
<td>$A^*-TOD$ previous best method faster by $9.0x$</td>
<td>$A^*-TOD$ previous best method faster by $8.0x$</td>
<td></td>
</tr>
</tbody>
</table>

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 $\phi$. $\epsilon$ 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 $\phi$ 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.

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


