Efficient Reduction of L-infinity Geometry Problems

Hongdong Li
Research School of Information Sciences and Engineering
Australian National University and NICTA
hongdong.li@anu.edu.au

Abstract

*This paper presents a new method for computing optimal L_∞ solutions for vision geometry problems, particularly for those problems of fixed-dimension and of large-scale. Our strategy for solving a large L_∞ problem is to reduce it to a finite set of smallest possible subproblems. By using the fact that many of the problems in question are pseudoconvex, we prove that such a reduction is possible. To actually solve these small subproblems efficiently, we propose a direct approach which makes no use of any convex optimizer (*e.g. *SOCP or LP), but is based on a simple local Newton method. We give both theoretic justification and experimental validation to the new method. Potentially, our new method can be made extremely fast.*

1. Introduction

The L_∞ optimization is a rather new and promising direction of research in multi-view geometry [4, 5, 6]. In just a few years' time it has attracted much attention from the vision geometry community (see *e.g*. [12, 17, 9, 14, 2, 7, 15]).

A critical virtue of the L_∞ scheme is that the solution obtained is not only geometrically meaningful, but also globally optimal and hence unique. The unique solution can always be reliably found, regardless of the size of the problem and the configurations of the cameras. This forms a sharp contrast to the conventional L_2 method, which is known to be problematic due to local minima or slow convergency.

This virtue entails the L_∞ scheme particularly suitable for large-scale applications. In these applications, there are often *e.g*. thousands of views to be triangulated, or thousands of points to be fit to a model. Paper [19] gives some instances of large problems, one of which, the 'Notre Dame' data set, involves solving for 595 cameras and 277,887 3D points.

However, to process large-scale data using L_∞ there is a *computational complexity* issue. The standard approach for solving an L_∞ problem is to convert it into a sequence of convex programs (*e.g*. SOCP, second-order cone program), and iteratively solves many such SOCPs via a bisection (binary search) procedure. But, convex optimization is anyway an expensive computation. Modern convex optimizers are mostly based on the Interior-Point-Algorithm, which is known to be *polynomial algorithm*, meaning that the required computation grows polynomially in the size of the problem (*i.e*. number of constraints and variables). A high polynomial degree often prevents it from handling too big problems, or its speed is unbearably slow.

To accelerate the L_∞ computation, researchers have explored various ways. However, most of them have been concentrated either on how to speedup the convex solver itself, or on how to reduce the total number of convex program iterations (*e.g*.,[5][2]).

As a consequence, despite having all those improvements, their methods still have to solve a full-size convex program using a full-scale convex optimizer at every iterations. Since convex optimizer has still polynomial complexity, also may come with a high memory (storage) requirement, it may not be capable of handling very large-scale problems.

The goal of this paper is to develop a new method to speed up the L_∞ computation, particularly for those large-scale applications. We will present a new approach that effectively solves an L_∞ problem *without* solving any convex program. Our method is not only theoretically provable, but also practically efficient. Potentially, it can be easily made extremely fast.

2. Related work

In an early work on L_∞ vision computation [5], Kahl formulated the problem as *quasi-convex* program, and proposed an *Improved Bisection* method based on the idea of updating the upper-bound adaptively, thus effectively reduces the total number of SOCP iterations. This way, considerable time reduction was observed.

A recent paper [2], also aiming at reducing the total number of iterations, has gained remarkable success. Based on *fractional programming*, it introduced five algorithms (*e.g*. Dinkelbach algorithm, Gugat algorithm, etc.) to vision community. The common idea is: instead of doing a naive binary search (bisection) which has linear convergence, it uses Newton method to cut-down the total number of SOCPs significantly.

Olsson *et al*. [12] realized that many L_∞ functions in vision are in fact *pseudoconvex* (which is slightly stronger than quasi-convex). They proposed two fast algorithms. The first one is based on local search using LOQO, but for large applications it suffers from numerical convergence problem. The second one is via SOCP approximation, and lately was shown to be a special case of Dinkelbach algorithm. Our work is significantly motivated by the concept of *pseudoconvexity*.

The new method, to be presented in this paper, is rooted from the mathematical theory of **LP-type problems**. In our previous work we introduced the elegant LP-type framework to multi-view L_∞ computation [7]. We have proven that many fixed-dimensional L_∞ geometry problems are in fact instances of the so-called LP-type problem [10]. However, that paper's focus was on outliers-removal, and no efficient algorithm was given there.

After the completion of the work, a very interesting connection was brought to our attention, which is that our method bears a remarkable similarity to the SMO method for fast Support Vector Machine training in the machine learning field [13]. We note that, while these two methods advocate the same idea of reducing to the smallest possible subproblems, the SMO was designed for the convex (quadratic programming) case, and the actual reduction algorithms are also different.

3. A Preview of the Paper

So far, almost all published works on L_∞ vision computation are based on convex optimization. As mentioned earlier, being a polynomial-complexity algorithm, the convex optimization is expensive, preventing it from handling very big problems.

For this reason, most existing wisdom for accelerating large-scale L_∞ computation is either through using faster convex solvers, or through reducing the total number of convex programs.

However, since convex-optimization appears to be the *bottleneck* here, *can we avoid it at all*? Motivated by this question, we take a very different angle to attack the problem of large-scale L_∞ computation. We ask ourselves a *novel* question: *is it possible to effectively solve an L_∞ problem without solving any convex program ?*

Our answer to this question is "yes." In this paper, we propose a new method precisely does this: enjoying all the benefits of L_∞ without using any (slow and expensive) convex optimizer.

Our method follows from the **reduction** (or decomposition) strategy, which performs by *reducing* a big original problem into a set of small-size subproblems. We call these small subproblems as *atom* or *primitive* problems. Each of the atom problems is so small that applying a full-scale convex solver is often unnecessary. Rather, more direct and more efficient methods (*e.g*. analytic) may exist and may suffice for solving them. This way, we avoid the use of convex optimizer.

To be able to apply the above strategy, it is crucial to show that the L_∞ problems are indeed "reducible". To this end, we will establish our **main reduction theorem** in **section-5**. The theorem is substantially grounded on the **pseudoconvexity** theory–which will be reviewed in **section-4**.

To actually perform the reduction (*i.e*. form the originally big problem to a set of small primitive problems), we present two **reduction algorithms** in **section-7**. These algorithms are efficient, in the sense that they will find the optimal solution in linear *expected* time. Therefore, our new method is also efficient.

Our *efficiency argument* also depends on the assumption that, one can solve those primitive problems extremely efficiently. This is often the case, in fact, as the primitive problems are usually very tiny and very regular. **Section-6** is devoted to **primitive solvers**.

Indeed, as an example, later we will show a 1000-point L_∞ planar homography problem can be reduced to 4-point homography and 5-point homography problems etc. Obviously, to estimate an L_∞ homography from 4 points one does not need any sophisticated SOCP; a simple SVD suffices.

4. The L_∞ Minimization and Pseudoconvexity

To ease exposition, this section will summarize some known results of L_∞ in vision computation. Emphasis is given to pseudoconvexity and its implications.

The key idea of the L_∞ scheme is to replace the L_2 error norm with the L_∞-norm (*i.e*. minimax norm). Previous works show that this leads to quasi-convex minimization (or *quasi-convex program*). We call a function *quasi-convex* if all of its sublevel sets are convex. Quasi-convex minimization in multi-view geometry often takes the following form ([5]):

$$\min_{\mathbf{x}} \ \max_{i} \ f_i(\mathbf{x}) = \frac{\|\mathtt{A}_i\mathbf{x}+\mathbf{b}_i\|}{(c_i^T\mathbf{x}+d_i)} \tag{1}$$

$$s.t. \quad c_i^T\mathbf{x}+d_i > 0, i = 1, ..., N, \tag{2}$$

where the $f_i(\mathbf{x})$ are quasi-convex, $\mathbf{x} \in \mathbb{R}^n$ is the unknowns to be solved for. The dimension of the problem is n, which is often fixed and intrinsic to particular application. For example, $n = 3$ for multi-view triangulation, $n = 6$ for 2D affinity, $n = 8$ for planar homography, $n = 7$ for fundamental matrix and $n = 11$ for camera calibration, etc. The size (scale) of the problem is defined by N. By 'large-scale' problems we mean $N \gg n$ in general.

The standard approach to solve such a quasi-convex program is to convert it to iteratively solving the following SOCPs via bisection:

$$\min_{\mathbf{x}} \ \gamma \tag{3}$$
$$s.t. \quad \mathcal{C}_i(\mathbf{x}) = \|\mathbb{A}_i\mathbf{x} + \mathbf{b}_i\| - \gamma(c_i^T\mathbf{x} + d_i) \leq 0, i = 1, ..., N,$$

where $\mathcal{C}_i(\mathbf{x})$ represents the i-th second-order cone (shell only). Note that we have multiplied the chirality condition to both side.

4.1. Useful results of pseudoconvexity

Paper [12] has proven that the above functions $f_i(\mathbf{x})$ are in fact *pseudoconvex* in the feasible region. Pseudoconvex is a slightly stronger condition than quasi-convex. A pseudoconvex function is always quasi-convex but the converse is not necessarily true.

Definition 4.1 *A function f is called* pseudoconvex *if it is differentiable and $\triangledown f(\bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}}) \geq 0$ implies $f(\mathbf{x}) \geq f(\bar{\mathbf{x}})$, where $\triangledown$ denotes gradient operator* [8].

Pseudoconvexity is very useful in minimization. Informally speaking, a local stationary point of a pseudoconvex function is also its global minimum. To find the global minimum it is sufficient to find a local stationary point, and this can be done *sufficiently* by solving a KKT system of the problem, as assured by the following two results.

Lemma 4.2 *Given a pseudoconvex function f, then we have $\triangledown f(\bar{\mathbf{x}}) = 0$* if and only if *$f(\mathbf{x}) \geq f(\bar{\mathbf{x}})$ for all $\mathbf{x}$.*

This lemma says that any *local stationary* point of a pseudoconvex function is also its global minimum.

Theorem 4.3 (KKT sufficient condition [8]) *Consider a general inequality-constrained minimization problem:*

$$\min_{\mathbf{x}} f(\mathbf{x}), \quad s.t. \quad g_i(\mathbf{x}) \leq 0, i = 1, ..., N \tag{4}$$
$$\mathbf{x} \ \textit{in a convex set}.$$

Let $\bar{\mathbf{x}}$ be a feasible solution. Suppose its KKT system holds true at $\bar{\mathbf{x}}$, i.e.*, there exist scalars $\lambda_i \geq 0$ such that*

$$\triangledown f(\bar{\mathbf{x}}) + \sum_i \lambda_i \triangledown g_i(\bar{\mathbf{x}}) = 0. \tag{5}$$

Suppose further that $f(\mathbf{x})$ and $g_i(\mathbf{x})$ are pseudoconvex, then the point $\bar{\mathbf{x}}$ which solves the KKT system is precisely the global optimum.

This theorem says that, unlike the quasi-convex case, for pseudoconvex case the KKT conditions are not only *necessary* but also *sufficient*. This actually suggests a local descent approach for pseudoconvex optimization. Following from this, paper [12] made an important attempt in using LOQO to solve L_∞ problems. For small problems LOQO worked fine, but for large problems it often failed to converge.

Minimax case. Recall that our purpose is to minimize $\max_i f_i(\mathbf{x})$, where each $f_i(\mathbf{x})$ is pseudoconvex. Although the concept of pseudoconvexity does *not* extend to the minimax case, we however have a significant result as follows:

Theorem 4.4 (minimax KKT) *$\mathbf{x}^*$ solves $\gamma^* = \min_{\mathbf{x}} \max_i f_i(\mathbf{x}), \ i = 1, ..., N$, where $f_i(\mathbf{x})$ is pseudoconvex,* if and only if *there exist scalars λ_i such that*

$$\sum_{i=1}^{N} \lambda_i \triangledown f_i(\mathbf{x}^*) = 0, \qquad \sum_{i=1}^{N} \lambda_i = 1, \tag{6}$$
$$\textit{where } \lambda_i \geq 0 \textit{ if } f_i(\mathbf{x}^*) = \gamma^*, \lambda_i = 0 \textit{ if } f_i(\mathbf{x}^*) < \gamma^*.$$

This theorem says that: at the optimum point $\mathbf{x}^*$ of γ, in each direction (denoted by θ) there must be an 'i' such that $\triangledown f_i(\mathbf{x}^*) \cdot \theta \geq$0. Geometrically, this means that in each direction there is at least one residual non-decreasing. Proofs of all the above results can be found in [8, 12].

5. Main Reduction Theorem

We have seen that pseudoconvexity is useful in deriving optimization procedures. In this section, we will further show that it actually offers more.

Recall that our intended strategy for solving a big problem of size N (c.f. Eq.3) is to reduce it to a single or a set of small primitive problems, each of which is of size $k \ll$N. By using pseudoconvexity, we will show this is indeed possible, thanks to the Main Reduction Theorem to be given below.

Before presenting the theorem, we first point to a useful empirical observation, which offers insight to the theorem. Many authors have observed that, the L_∞ optimum is actually only *supported* by a small subset of the entire constraints set. This empirical 'fact' was applied to L_∞ computations (*e.g.* [18, 15, 7]). But neither of them has offered sufficient justification, nor answered a key question: *"how big size that subset should be, and how to quickly find it?".*

5.1. The Main Theorem

We now state the main reduction theorem.

Theorem 5.1 (main reduction theorem)
Consider L_∞ problem $\min_{\mathbf{x}} \max_i f_i(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, $i = 1, ..., N$, $N > (n+1)$, each $f_i(\mathbf{x})$ is pseudoconvex. Denote $f_{\mathtt{I}}(\mathbf{x}) = \max_{i \in \mathtt{I}} f_i(\mathbf{x})$, where $\mathtt{I}$ is a subset of $\mathtt{N} = \{1, ..., N\}$. Also denote $f^*_{\mathtt{I}}$ as the minimum of $f_{\mathtt{I}}(\mathbf{x})$. Then there must exist a proper subset $\mathtt{B}$ which is of size $|B| \leq n+1$ such that $f^*_{\mathtt{B}} = f^*_{\mathtt{N}}$ and $x^*_{\mathtt{B}} = x^*_{\mathtt{N}}$ are their (equal) optimizers.

We call a subset of $\mathtt{N}$ a *basis set* (or *basis* in short), *if* this subset yields the same minimal function value *and if* remove any member from it will decrease the function value. The maximal size (cardinality) of any basis is called the *combinatorial dimension*. Hence the combinatorial dimension in the theorem is $(n+1)$.

A proof of the theorem is given in the Appendix of the paper. It is adduced in a form most suitable for the purpose. A related and neat result, though in a different context, can be found in [11].

This theorem predicts that the solution of the big problem is dominated by a small basis set of at most $(n+1)$ constraints. This is very appealing, because solving that small sub-problem yields identically the same result to the original problem. Now the remaining task is to *quickly find a basis*.

If the purpose were merely *to find a basis*, then one could easily do it *a posteriori*, meaning that extracting a basis set after having found a solution of the original problem. But this is meaningless however, for our current purpose of reduction. We must have an *a priori* approach that can identify a basis quickly before spending too much time on solving the original big problem. But how ?

Our answer to this will be given in section-7, in which we will present two reduction algorithms, both of which can find a basis in linear expected time. In the next section (*i.e.*, section-6), instead, we will explain how to construct primitive solvers.

6. Solving Primitive Problems

In this section, we will elaborate on how to actually compute the primitive problems fast and efficiently, without using convex optimizer. We choose to describe our primitive solvers earlier (before describing the actual reduction algorithms in the next section), is to assure the reader earlier of the possibility of *solving L_∞ without SOCP*.

Let us use **N-view triangulation** as an example to illustrate the possibility to construct *ad hoc* primitive solvers without SOCP. The combinatorial dimension here is 4. Hence only 4 distinct primitive problems exist: 1-view, 2-view, 3-view and 4-view (the 1-view case is trivial). Note that they are all under the L_∞ norm.

Suppose by using our reduction algorithms (to be given in the next section) we have reduced a big N-view problem into a small set of k-view ($k \leq 4$) primitive problems. Because the primitive problems are so small that one is able to solve them *directly*, efficiently, or even in closed-form. Now let us show this.

2-view case. It is easy to imagine (via geometric intuition) that, solving 2-view L_∞ triangulation is equivalent to finding two non-inclusive cones that are tangent in 3-space. The optimum attains, *i.e.*, the γ values is minimal, when the two cones are tangent with the two gradients at the point of tangency pointing to opposite directions. This point-of-tangency gives the optimal solution. Summarizing this up mathematically and we get:

$$\begin{cases} \mathcal{C}_i(\mathbf{x}) = \|\mathbb{A}_i\mathbf{x} + \mathbf{b}_i\| - \gamma\,(c_i^T\mathbf{x} + d_i) = 0, \;\; i = 1, 2 \\ \lambda \triangledown \mathcal{C}_1(\mathbf{x}) + (1-\lambda)\triangledown \mathcal{C}_2(\mathbf{x}) = 0, \;\; \lambda > 0, \text{a scalar.} \end{cases}$$

This is almost a system of equations. Having very small size, the system can be easily solved by any Newton method (*e.g.* Levenberg-Marquardt). In solving the system, the last positivity condition can be relaxed first and reinforced afterward. A linear triangulation method may be used to provide an initial point. To ensure that the linear solution is also feasible in terms of chirality, a validation before using it seems necessary.

The reader may worry that the local Newton method may not converge globally. The fact is, being pseudoconvex the problem has only one local minimum, which is also the global optimum. In addition, because the problem size is so tiny and the nonlinearity is so mild that the Newton solver is adequate in practice. Our tests hardly encountered any difficulties, even when the noise level was high.

3-view case. The 3-view case is a bit tricky. Now the 3 cones cannot be tangent in any configuration, otherwise it will reduce to the 2-view case, contradict to the fact that the 3 cones form a *basis* set. Hence, they must intersect at a common point. Moreover, at that point the three gradients must cover all feasible directions, otherwise the point would move therefore not non-optimal, contradict to the preassumption. Mathematically we have:

$$\begin{cases} \mathcal{C}_i(\mathbf{x}) = \|\mathbb{A}_i\mathbf{x} + \mathbf{b}_i\| - \gamma\,(c_i^T\mathbf{x} + d_i) = 0, \; i = 1, ..., 3 \\ \qquad \sum_i \lambda_i \triangledown \mathcal{C}_i(\mathbf{x}) = 0, \;\; \sum_i \lambda_i = 1, \; \lambda_i > 0. \end{cases}$$

4-view case. The 4-view case, on the other hand, is particularly simple. Note that we have totally 4 unknowns to solve, 3 in $\mathbf{x}$ and 1 in γ. Now we have precisely 4 cones in the basis set. Hence the solution can be found as a proper root of the square system of equalities.

$$\mathcal{C}_i(\mathbf{x}) = \|\mathbb{A}_i\mathbf{x} + \mathbf{b}_i\| - \gamma\,(c_i^T\mathbf{x} + d_i) = 0, \; i = 1, ..., 4$$

There may be multiple roots. However, this is no serious, thanks to the use of local solver, and to the fact that we can easily rule out the spurious roots by examining the 4 gradients at the root, as in the 3-cone case.

Extensions. (i.) The above direct solvers are specifically designed for triangulation. But, similar procedures can be derived easily for other problems as well. Moreover, a common pattern for their computations exists: Consider a problem with degree-of-freedom d and combinatorial dimension $(n+1)$. We want to find its k-cone solution. Then we have: the case of $k < d$ is trivial; the case of k=d is nothing but the minimal case which can be solved linearly; the case of k =n+1 leads to a *square* system of equations; the cases of $d < k \leq n$ may be solved by examining the k gradients. (ii.) The foregoing direct solvers work well in practice. However, the way to obtain them is rather *ad hoc* and *heuristic*. One can imagine, it would soon become tedious as the problem's combinatorial dimension grows higher. However, by using theorem-4.4 it is not hard to find a systematic way to construct *generic* solvers.

7. Two Reduction Algorithms

By far all other preparations are ready. One key question remains: how to find a basis quickly ?

Using the combinatorial dimension bound $\delta = (n+1)$, one could naïvely enumerate all possible subsets and choose the one that yields the smallest values as a basis. For a size N problem, this gives a plausible algorithm but of complexity O$\big(N^1 + ... + N^{n+1}\big)$, which is impractical to use.

We will introduce two practical reduction algorithms in this section. The algorithms accept as input the pseudoconvex program **P1** (Eq.3) with N constraints, reduce it to a set of primitive problems of size at most $(n+1)$, and finally output a single basis within finitely-many steps. These two algorithms are in fact equivalent to each other. They differ only in the forms of coding or implementations: one of them is *recursive*, and the other is *iterative*. The recursive version is neat and element, easy to analyze, while the iterative version performs faster on certain platform as it avoids much of the function-calls overhead. We call our algorithms *QuickPseudo Algorithm.*

7.1. The recursive algorithm

The actual algorithm for recursive-reduction is summarized in Algorithm-1. In it, we say a constraint h *violates* a set B, if adding h to B leads to an increase in the function value. Easy to check that the condition $|B| < \delta$ is redundant; it is nevertheless kept here for ease presentation.

The **correctness of the algorithm** can be verified by induction. Because there are only two recursive calls inside the recursive function, both of them *decrease* the size of P while maintain and *increase* the size of B in a range of $0 \leq |B| \leq \delta$. Therefore, the algorithm will always terminate in finite steps, and the output will be a basis, since it must pass all the *violation* tests.

```
Input: Problem P1 (Eq.3), constraints set N, combinatorial
       dimension $\delta$
Output: a single basis B of Problem P1
begin
    Initialization: Store N in a list and then perform a
    random permutation $\pi$ on it, i.e., N := $\pi$(N);
    return QuickPseudo($N, \varnothing$);
end

  Function QuickPseudo($P, B$)
  begin
      % P is a list of constraints, B is an initial basis ;
      if ($|B| < \delta$ and P = $\varnothing$) or $|B| = \delta$ then
          $\gamma$:= PrimitiveSolver($B$);
          D := B ;
      else
          choose the last $p \in$ P ;
          D:=QuickPseudo($\{P \backslash p\}, B$) ;
          if $p$ violates $D$ then
              D := QuickPseudo($\{P \backslash p\}, \{B \cup p\}$) ;
              move $p$ to the front of the list P ;
          end
      end
      return $D$ ;
  end
```

Algorithm 1: Recursive QuickPseudo Algorithm

7.2. The iterative algorithm

```
Input: Problem P1, the constraint set N
Output: a single basis B of P1
begin
    N := $\pi$(N) ;
    B := $\varnothing$ ;
    repeat
        % test elements of N one after another;
        if $p \in$ N violates B then
            B := BasisUpdate($B, p$) ;
            move $p$ to the front of the list N ;
        end
    until no $p \in$ N violates B ;
    return B ;
end
```

Algorithm 2: Iterative QuickPseudo Algorithm

The analysis to this algorithm is similar to the above one. The internal function BasisUpdate is used to find the basis of an enlarged set consisting of the old basis B and a violating constraint p. The algorithm starts from B $= \varnothing$. Based on its functionality, details of the function BasisUpdate can be easily filled in by the reader, thus is left out for space reason. The step of "move p to the front of the list" in both algorithms, is based on the well-known *move-to-front* heuristics in algorithm research. It is used to "pivot" important solutions, so that the algorithms run practically faster.

Complexity Analysis. The proposed two algorithms are special cases of the LP-type algorithm in the LP-type theory. The reader is referred to [16, 10] and [3] for excellent references with detailed analysis of the algorithms.

Both algorithms are *randomized algorithms*, as seen by the fact that a random permutation is used to initialize them. Despite being randomized algorithms, on average their *expected* computational time is fixed, and a careful counting of the algorithm steps reveals that, both the algorithms have an expected linear computational complexity,*i.e. linear* in the problem size $O(\xi|\mathtt{N}|)$, where ξ is a constant depending on the problem's combinatorial dimension.

The implication of such a linear complexity is mixed: on one hand, one can quickly (on average) find a basis after a linear number of primitive computations; On the other hand, the hidden constant ξ can be very big if the problem dimension is high, because it depends upon the dimension up to (sub)exponential (*e.g.*, a rough estimate is that $\xi \approx \delta!$). Due to such a (sub)exponential bound, the overall efficiency of these algorithms is much compromised.

So far, in theory there is still no exact *cure* to the big ξ problem. However, in practice: (1) many geometry problems are of low dimension in nature; (2) for high-dimension problems one may count on some super-fast implementations of the primitive solvers; (3) thirdly, if one settle for some *approximate solutions*, then the order of dependency of ξ on the problem dimension may be reduced substantially. For example, a desirable case would be that the ξ depends on δ only linearly (or quadratically, or cubically, ...), rather than (sub)exponentially.

Following the third point (of approximate optimal solution), we now point out a possible (and seemingly promising) remedy to the big- ξ problem, that is the **Core-Set** paradigm. The 'Core-Set' is a new and significant idea being developed recently in the field of computational geometry [1]. Its key motivation is to find an *approximate* optimal solution to the original problem by using a very small (and often sparse) subset (called a core-set) of the data, form which an approximate ε-optimal solution can be found (ε being an accuracy factor). A fundamental property of such core-set approximation is that, for certain problems there are ε-core-sets whose size depend only on ε, but not on the problem dimension (δ), or on the problem size ($|\mathtt{N}|$). If such core-sets do exist for our L_∞ multiview geometry problems, then potentially a linearly-bounded linear-time algorithm may be constructed. We believe this is an interesting future research direction.

8. Experiments

We tested the proposed method on both synthetic data and real data. For synthetic data, we simulated 1,000 point clouds within a cube in front of all N views. The synthetic images' sizes are roughly of 1000×1000 and uniformly random noise of up to ± 5 pixels were added in.

Both the recursive version and the iterative version of our reduction algorithm have been implemented and tested. They show no difference in outputs, but the latter one is slightly fast. This is in accord with the common argument w.r.t. iteration vs. recursion. So, in below our timing reports are based on the iterative version. The test environment is consisted of a standard PC (P4-3G, 1GB 32bits) and Matlab 2008a. We use Matlab's `profiler` to report the timings and comparisons. The adopted SOCP solver is SEDUMI.

Method:Time(seconds)	2-view	3-view	4-view
Improved Bisection	0.58	0.71	0.80
Our direct solvers	0.010	0.012	0.014

Table 1. Primitive solvers: time comparison.

The base-line algorithm used for comparison is the *improved bisection* of [5], which is the best implementation of the bisection SOCP approach, and it is already much faster than the standard bisection procedure. We understand that such comparisons are qualitative only, as the results are much dependent of the actual implementation. Comparing absolute running timing with Matlab code gives merely a relative performance index. In particular, from a reduction algorithm's point of view, a better question to answer is that: *given a big pseudoconvex program, how many primitive problems do we need to solve, and how fast can we solve each of them ?*

In deed, a central argument of the paper is: if one can provide super-fast, or closed-form, or hardware parallelized primitive solvers, then the overall performance of our new method will be remarkably high.

8.1. Primitive triangulation

We use Matlab's `fsolve` (with levenberg marquardt option and user-provided analytic jacobians) as our primitive solvers (see section-6). Initial points for `fsolve` are found via linear method. The theory of pseudoconvexity guarantees that a local solution is also the global optimum. The size of primitive problem is so small that the Levenberg-Marquardt algorithm always converges practically.

Two methods, *i.e.*, the Improved-Bisection and our primitive solvers, are tested on the 2-view, 3-view, 4-view cases. Both methods are set to terminate at the same accuracy (1e-5), and their final results are verified to be identical (up to the accuracy).

The following results (table-1, average of 100 runs) clearly show that our specially-designed primitive solvers perform much faster than the Improved-Bisection. On average, with our current un-optimized Matlab code we can solve a primitive problem in about 10 milliseconds. We believe it can be reduced to the order of 1 milliseconds, if much optimized C code is to be used.

Problems	I-Bisection	Our method	#(prim-prob)
10-view	1.02s	0.19s	6
100-view	1.25s	0.82s	17
1000-view	6.30s	3.31s	33
10000-view	92.10s	6.02s	49
20000-view	310.12s	11.8s	65
50000-view	1520s	15.1s	77

Table 2. Reduction results. Left two columns: core computation time; Right column: the numbers of resulting primitive problems.

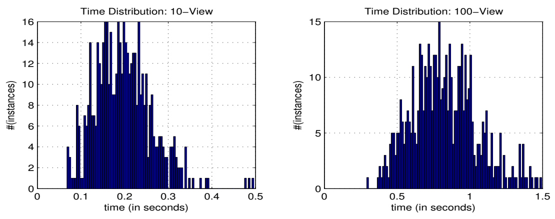

Figure 1. Histogram of running time (over 500 tests) for "10-view triangulation" (left) and "100-view triangulation" (right) by our new method.

8.2. N-view triangulation

Then we tested the iterative reduction algorithm on N-view triangulation. We deliberately made N very big, just to confirm the performance of our reduction under extreme cases.

Table-2 gives comparison of core timings and the number of resulting primitive problems (round to integers). These results are also the average of 100 runs (up to the 10,000-view case). The reported times are core computational time including those spent on reductions and on primitive problems. The actual numbers of primitive problems are much smaller than that theoretically-predicted. We believe this phenomenon is owe to the peculiar nature of triangulation.

Recall that our algorithm is randomized algorithm, therefore we would like to know the variations of execution times. Two sample results are provided in fig-1.

N-point homography. We also tested our method on N-point 2D affine problem, N-point planar homography problem and N-point camera resectioning problem. The combinatorial dimensions of these problems are higher than triangulation. Our method still runs correctly, but the improvement is less evident, or even much slower for large N.

However, as we have always argued, our method can be made fast, as long as one can implement very fast primitive solvers. This is indeed possible in practice. Since the sizes of these primitive problems are so tiny and very regular, they are very suitable for hardware or parallel computation.

Data Sets	I-Bisection	Our method	#(prim-prob)
Dinosaur (full)	3676s	365s	8897
NotreDame (1/10)	35815s≈10h	4968s≈ $1\frac{1}{2}$h	152780

Table 3. Real test results: the dinosaur data set and 1-tenth of the Notre Dame data set.

We envisage the real potentials of applying GPU to the L_∞ computation. For example, it takes little time to compute a batch of 5-point homographies in parallel on GPU.

8.3. Tests on real images

We validated our method on two real data sets, one is the well-known Dinosaur sequences (fig-2), the other is the Notre Dame data courtesy of [19] (fig-3). The Dinosaur contains 36 view of 4,983 tracks and 16,432 feature points. The Notre Dame contains 595 views of 277,877 tracks and about one million feature points. We only tested 1-tenth of tracks in the Notre Dame data set, which is probably already sufficient to give a performance indication.

For the Dinosaur data set (4983 points in full) our method spent 6 minutes in getting an L_∞ reconstruction, while the Improved-Bisection took more than one hour. In terms of reduction, the original problem has been reduced to 8,897 small primitive problems. For the Notre-Dame data set, the improvement is even more apparent, as shown in table-3.

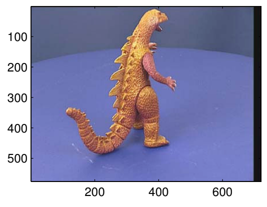
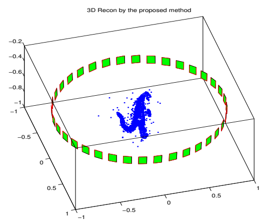

Figure 2. Dinosaur Sequence: a sample image and 3D recon.

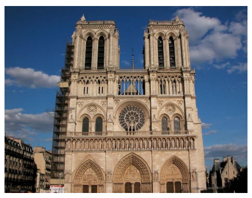
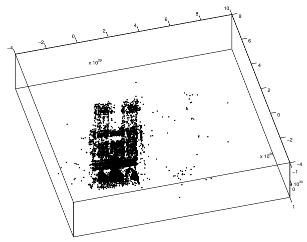

Figure 3. Notre Dame dataset: a sample image and 3D recon.

9. Discussion and Conclusion

From the above analysis we can see, the proposed method is most suitable for problems having low-dimension (*i.e.*, a few variables) but subjecting to a large number of constraints (*i.e.* the constraints-to-variables ratio is high). If this is the case, our method performs the best.

Moreover, the method is only effective for problems having fixed and low dimensions. In the area of multi-view geometry, however, there are problems not falling into this category. For example, structure-and-motion from known rotation (dimension is not fixed) or plane-based reconstruction (high and varying dimension) are two exceptions. For these problems, a better solution is [2].

So far, for medium-dimension problems the *absolute speedup* by our method is not significant. However, the potential to reach faster speed is high. Not like the original big problem, the primitive problems are so tiny and regular. Therefore, finding a closed-form solution (via *e.g.* Gröbner basis), or a hardware (*e.g.* GPU) implementation are both possible.

To conclude, if the optimal solution of a big problem is dominated by a small subproblem, then why bother solving the big one? The present work offers a valuable way to efficiently solve large-scale L_∞ multi-view geometry problems. We hope this paper will contribute to the endeavor of applying the elegant L_∞ scheme to large-scale real-world applications.

Acknowledgement We thank all three anonymous reviewers' positive comments and suggestions. We would like to thank Carl Olsson (Lund) for his invaluable remarks and for the simplified proof of the main reduction theorem. The research is supported in part by CRL NICTA (the VIBE project). NICTA is a national ICT research centre funded by the Australian Government, Department of Communications, Information Technology and the Arts.

References

[1] P. Agarwal, S. Har-Peled, and K. Varadarajan. Geometric approximations via coresets. *Combinatorial and Computational Geometry –MSRI Publications*, pages 1–30, 2005.

[2] S. Agarwal, N. Snavely, and S. Seitz. Fast algorithms for linfty problems in multiview geometry. *In Proc. CVPR*, June 2008.

[3] J. E. Goodman and J. O'Rourke, editors. *Handbook of discrete and computational geometry*. CRC Press, Inc., 1997.

[4] R. Hartley and F. Schaffalitzky. l_∞ minimization in geometric reconstruction problems. *In Proc. CVPR*, 1:504–509, 2004.

[5] F. Kahl. Multiple view geometry and the l_∞-norm. *In Proc.ICCV*, 2005.

[6] Q. Ke and T. Kanade. Quasiconvex optimization for robust geometric reconstruction. *In Proc.ICCV*, pages 986–993, 2005.

[7] H. Li. A practical algorithm for l∞ triangulation with outliers. *In Proc. CVPR*, June 2007.

[8] O. Mangasarian. *Nonlinear programming*. SIAM, 1994.

[9] D. Martinec and T. Pajdla. Robust rotation and translation estimation in multiview reconstruction. *In Proc. CVPR*, 2007.

[10] J. Matousek, M. Sharir, and E.Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498, 1996.

[11] C. Olsson, O. Enqvist, and F. Kahl. A polynomial-time bound for matching and registration with outliers. *In Proc. CVPR*, 2008.

[12] C. Olsson, A. Eriksson, and F. Kahl. Efficient optimization for l_∞ problems using pseudoconvexity. *In Proc. ICCV*, 2007.

[13] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods: Support Vector Machines*, 1998.

[14] M. Salzmann, R. Hartley, and P. Fua. Convex optimization for deformable surface 3-d tracking. *In Proc. ICCV*, Oct. 2007.

[15] Y. Seo and R. Hartley. A fast method to minimize l-infty error norm for geometric vision problems. *In Proc. ICCV*, Oct. 2007.

[16] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. *In Proc. STACS: Theoretical Aspects of Computer Science*, pages 569–579, 1992.

[17] K. Sim and R. Hartley. Recovering camera motion using l_∞ minimization. *In Proc. CVPR*, 1:1230–1237, 2006.

[18] K. Sim and R. Hartley. Removing outliers using the l_∞ norm. *In Proc. CVPR*, 1:485–494, 2006.

[19] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. *SIGGRAPH*, pages 835–846, 2006.

A. Appendix: proof of the main reduction theorem in section-5

To prove the theorem, we will need Helly's theorem—a well-known result in geometry and in convex optimization.

Theorem A.1 *(Helly's theorem) Given a set of N convex objects in n-dimensional space $\mathbb{R}^n$, $N >$ n+1, if* each *(n+1) objects has a common point, then there exists a point that is common to all the N objects.*

Recall $f_i : \mathbb{R}^n \mapsto \mathbb{R}_+$ is psudoconvex. Assume that the sublevelsets

$$S_i(\mu) = \{x \in \mathbb{R}^n; f_i(x) \le \mu\}$$

are compact. We first form all subsets of size $(n+1)$ and order them into $I_1, I_2, ..., I_M$, where $M \ge n+1$. We let

$$S_{I_j}(\mu) = \{x \in \mathbb{R}^n; f_i(x) \le \mu, \forall i \in I_i\} = \bigcap_{i \in I_j} S_i(\mu)$$

and

$$f^*_{I_j} = \min_x \max_{i \in I_j} f_i(x)$$

Then $f^*_{I_j} \le f^*_N$. Let

$$\mu^* = \max(f^*_{I_1}, f^*_{I_2}, ..., f^*_{I_N})..$$

Since for any fixed j, $S_{I_j}(\mu^*)$ is non-empty we have that all $n+1$ sets $S_i(\mu^*)$, $i \in I_j$ have a point in common. Therefore according to Helly's theorem there is a point $x^* \in \bigcap_{i \in I_N} S_i(\mu^*)$. It's easy to see that $\mu^* = f^*_N$. For the point x^* we therefore have that $f^*_N = \max_{i \in N} f_i(x^*)$. Finally, to show that $\mu^* = \max f^*_{I_j}$ for some j we note that, as we have seen there is j such that

$$\mu^* = \min_x \max_{I_j} f_j(x) \le \max_{I_j} f_j(x^*)$$

But since $x \in \bigcap_{i \in I_N} S_i(\mu^*)$ we also have

$$\max_{I_j} f_j(x^*) \le \mu^*$$

And therefore x^* and $B = I_j$ for this j works. □