Certifying Algorithms

Kurt Mehlhorn

MPI für Informatik
Saarbrücken
Germany
Outline of Talk

• run uncertified planarity test

• Certifying Algorithms: motivation and concept

• Three examples:
  • testing bipartiteness of graphs
  • testing planarity of graphs, run slow and fast uncertified planarity test
  • convex hulls in arbitrary dimensions

• Advantages of certifying programs

• Deterministic programs can be made certifying

• General strategies for finding certifying algorithms
  • Linear programming duality, run matching demo
  • Characterization theorems

• Programs with preconditions

• Certification and verification
The Problem Statement

- a user feeds $x$ to the program, the program returns $y$.
- how can the user be sure that, indeed, $y = f(x)$.
- the user has no way to know,
  - he must trust the program and its author,
  - he is at complete mercy of the program
- I do not like to depend on software in this way,
  not even for programs written by myself
Warning Examples

- LEDA 2.0 planarity test was incorrect

- Rhino3d (a CAD systems) fails to compute correct intersection of two cyclinders and two spheres

- CPLEX (a linear programming solver) fails on benchmark problem *etamacro*.

- Mathematica 4.2 (a mathematics systems) fails to solve a small integer linear program

```math
In[1] := ConstrainedMin[ x , {x==1,x==2} , {x} ]
Out[1] = {2, {x->2}}

In[1] := ConstrainedMax[ x , {x==1,x==2} , {x} ]
ConstrainedMax::lpsub": The problem is unbounded."
Out[2] = {Infinity, {x -> Indeterminate}}
```
programs must justify (prove) their answers in a way that is easily checked by their users.
Certifying Algorithms

- A certifying program returns the function value $y$ and a certificate (witness) $w$.

- $w$ proves the equality $y = f(x)$ even to a dummy.

- And there is a simple program $C$, the *checker*, that verifies the validity of the proof.

- Many programs in LEDA system are certifying.

- Name introduced in Kratsch/McConnell/Mehlhorn/Spinrad: SODA 2003.

- Related work: Blum et al.: Programs that check their work.
Testing Bipartiteness of a Graph

- A graph is bipartite if it can be two-colored.

- Algorithm
  - construct a spanning tree and color it
  - check all non-tree edges \( e = (v, w) \)
    - if \( v \) and \( w \) have different colors, continue
    - if \( v \) and \( w \) have the same color, stop and declare \( G \) non-bipartite

- How can we make this algorithm certifying?
Testing Bipartiteness of a Graph

- A graph is bipartite if it can be two-colored.

- Algorithm
  - construct a spanning tree and color it
  - check all non-tree edges \( e = (v, w) \)
    - if \( v \) and \( w \) have different colors, continue
    - if \( v \) and \( w \) have the same color, stop and declare \( G \) non-bipartite

- How can we make this algorithm certifying?

- extract an odd cycle in the case of failure?
Planarity Testing

- given a graph $G$, decide whether it is planar
- Tarjan (76): planarity can be tested in linear time
- a story and a demo
- combinatorial planar embedding is a witness for planarity
- Chiba et al (85): planar embedding of a planar $G$ in linear time
- Kuratowski subgraph is a witness for non-planarity
- Hundack/M/Näher (97): Kuratowski subgraph of non-planar $G$ in linear time

$K_5$ and $K_{3,3}$
**Convex Hulls**

Given a simplicial, piecewise linear closed hyper-surface $F$ in $d$-space decide whether $F$ is the surface of a convex polytope.

**FACT:** $F$ is convex iff it passes the following three tests

1. check local convexity at every ridge
2. $0 = \text{center of gravity of all vertices}$
   check whether $0$ is on the negative side of all facets
3. $p = \text{center of gravity of vertices of some facet } f$
   check whether ray $\overrightarrow{0p}$ intersects closure of facet different from $f$
Sufficiency of Test is a Non-Trivial Claim

- ray for third test cannot be chosen arbitrarily, since in $R^d$, $d \geq 3$, ray may “escape” through lower-dimensional feature.
The Advantages of Certifying Algorithms

- certifying algs can be tested on
  - every input
  - and not just on inputs for which the result is known.

- certifying programs are reliable
  - either give the correct answer
  - or notice that they have erred

- trustless computing
  - there is no need to understand the program, understanding the witness property and the checking program suffices.
  - one may even keep the program secret and only publish the checker

- formal verification of checkers is feasible

- most programs in LEDA are certifying
Does every Function have a Certifying Alg?

- formalize the notion of a certifying algorithm
  - let $P$ be a program and let $f$ be the function computed by $P$
  - a program $Q$ is a certifying program for $f$ if there is a predicate $W$ such that
    1. $W$ is a witness predicate for $f$.
      - $\forall x, y \ (\exists w \ W(x, y, w))$ iff $(y = f(x))$.
      - given $x$, $y$, and $w$, it is trivial to decide if $W(x, y, w)$ holds
    2. On input $x$, $Q$ computes a triple $(x, y, w)$ with $W(x, y, w)$.
    3. the resource consumption (time, space) of $Q$ on $x$ is at most a constant factor larger than the resource consumption of $P$

- "Theorem": Every deterministic program can be made certifying.
  - Proof: witness = correctness proof in some formal system

- construction is reassuring, but unnatural. The challenge is to find natural certifying algs
Design of Certifying Algorithms

- general approaches
- characterization theorem, e.g.,
  - non-planarity and Kuratowski subgraphs,
  - convex bodies and certifying rays
  - Delaunay triangulations and empty ball property
- linear programming duality: primal and dual solution certify each other:

\[
\max c^T x \quad \text{s.t.} \quad Ax \leq b, x \geq 0 \quad = \quad \min y^T b \quad \text{s.t.} \quad y^T A \geq c, y \geq 0
\]

- matchings and covers,
- flows and cuts,
- shortest paths and potential functions

- however, there is no “Königsweg”
Bipartite and General Matching

- given a bipartite graph, compute a maximum matching
- a matching $M$ is a set of edges no two of which share an endpoint
- a node cover $C$ is a set of nodes such that every edge of $G$ is incident to some node in $C$.
- $|M| \leq |C|$ for any matching $M$ and any node cover $C$.
  - map $(u, v) \in M$ to an endpoint in $C$, this is possible and injective

- a certifying alg returns $M$ and $C$ with $|M| = |C|$
- no need to understand that such a $C$ exists (!!!)
- it suffices to understand the inequality $|M| \leq |C|$
- demo for general graphs
Programs with Preconditions

- on input $x$ having property $\phi(x)$ compute a $y$ satisfying $\psi(x, y)$.
- $\phi$ is called precondition, $\psi$ is called postcondition
- strong or weak certifying algorithms
  - strong: produce a witness $w$ that either proves $\neg \phi(x)$ or proves $\psi(x, y)$ and tells us which
  - weak: produce a witness $w$ that proves $\neg \phi(x) \lor \psi(x, y)$

- can programs always be made weakly certifying?
- are weak and strong certifying algs the right notions?
A Weakly Certifying Version of Binary Search

• given a sorted array $A[1, n]$ and an input $x$ decide whether $x = A[i]$ for some $i$.

• run binary search on $A$, this will either determine an $i$ with $x = A[i]$ or an $i$ with $A[i] < x < A[i+1]$
  • in the former case return (yes,$i$)
  • in the latter case return (no,$i$)

• the former witness proves $x \in A$

• if the array is sorted, the latter witness proves $x \not\in A$
Example of a Strongly Certifying Algorithm

- input = a planar graph $G$, output = a five-coloring of $G$
- algorithm either proves that $G$ is non-planar or five-colors it
  - if $G$ has no vertex of degree 5 or less, declare $G$ non-planar
  - let $v$ be such a vertex
    - degree four or less: remove, color $G \setminus v$, add $v$ and color it
    - degree five: if the neighbors form a $K_5$ declare $G$ non-planar
    - otherwise there are two neighbors $u$ and $w$ which are not connected by an edge. Remove $v$ and identify $u$ and $w$. Color the reduced graph, split $u$ and $w$ (they use the same color), put $v$ back it and color it with a color not used on its neighbors.
Verification of Checkers

- the checker should be so simple that its correctness is “obvious”.
- we may hope to formally verify the correctness of the implementation of the checker
  this is a much simpler task than verifying the solution algorithm
    - the mathematics required for the checker is usually much simpler that the one underlying the algorithm for finding solutions and witnesses
    - checkers are simple programs
    - algorithmicists may be willing to code the checkers in languages which ease verification
    - logicians may be willing to verify the checkers

- **Remark:** for a correct program, verification of the checker is as good as verification of the program itself
- are exploring the idea together with programming logics group
Cooperation of Verification and Checking

- a sorting routine working on a set $S$
  - (a) must not change $S$ and
  - (b) must produce a sorted output.
- I learned the example from Gerhard Goos
- the first property is hard to check (provably as hard as sorting)
- but usually trivial to prove, e.g.,
  if the sorting algorithm uses a \textit{swap}-subroutine to exchange items.
- the second property is easy to check by a linear scan over the output,
  but hard to prove (if the sorting algorithm is complex).
- give other examples where a combination of verification and checking does the job
Summary

• certifying algs have many advantages over standard algs
  • can be tested on every input
  • can assumed to be reliable
  • can be relied on without knowing code
  • are a way to trustless computing

• they exist: every deterministic alg has a certifying counterpart

• they are non-trivial to find

• most programs in the LEDA system are certifying

• Monte Carlo algs resist certification

When you design your next algorithm, make it certifying