Set-Up

set up non-certifying and certifying planarity demo. Let the non-certifying demo run during introduction

Certifying Algorithms

Algorithms meet Software Engineering

Kurt Mehlhorn

MPI for Informatics and Saarland University

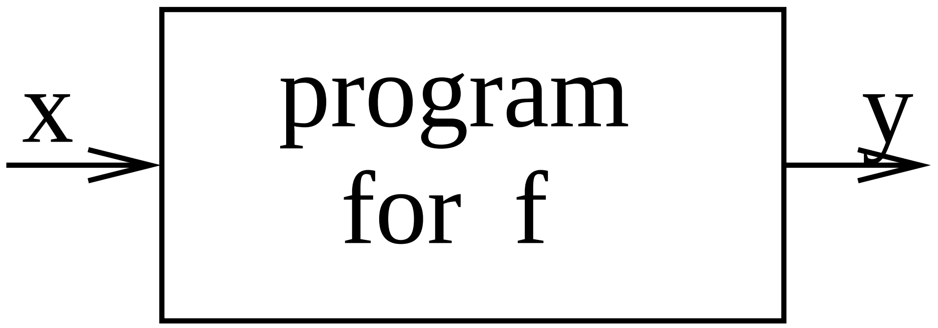
Saarbrücken

Germany

Outline of Talk

- Part I: Certifying Algorithms: An Overview
- Part II: Current Projects

The Problem



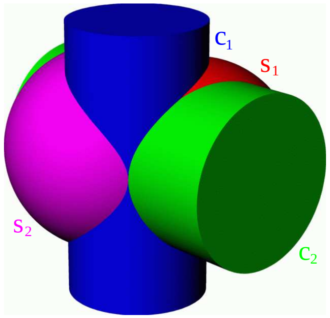
- A user feeds x to the program, the program returns y.
- How can the user be sure that, indeed,

$$y = f(x)?$$

The user has no way to know.

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

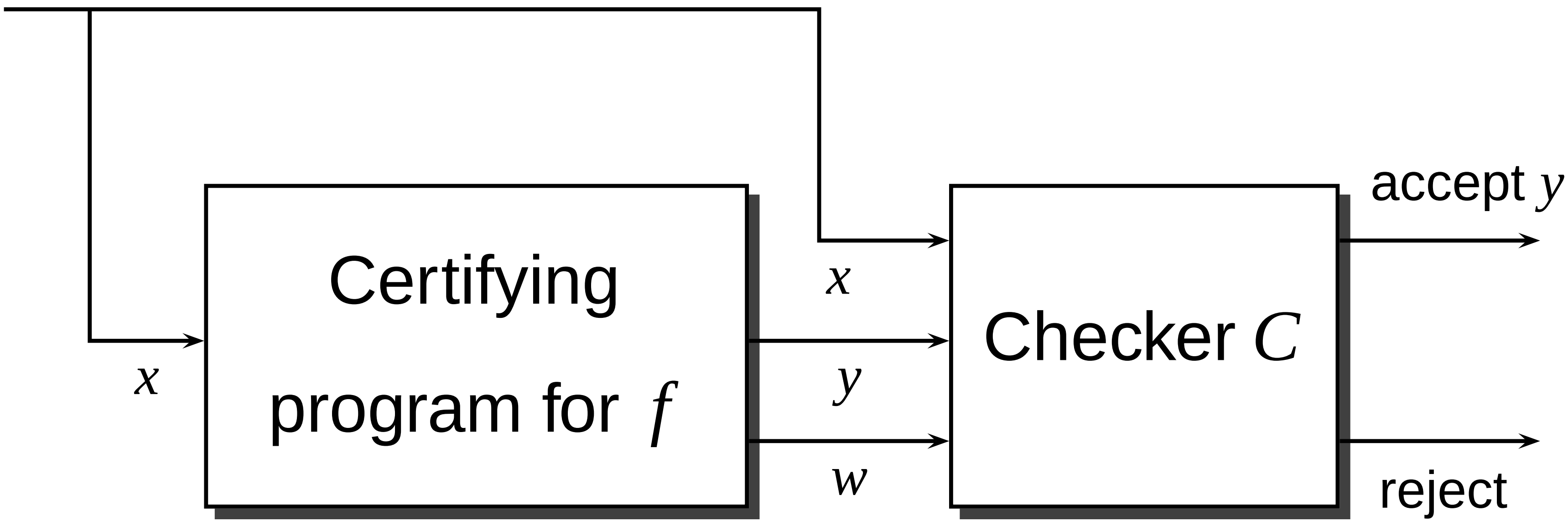
In[1] := ConstrainedMin[x , {x==1,x==2} , {x}]
Out[1] = {2, {x->2}}

In[1] := ConstrainedMax[x , {x==1,x==2} , {x}]
ConstrainedMax::lpsub": The problem is unbounded."
Out[2] = {Infinity, {x -> Indeterminate}}

The Proposal

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 $y = f(x)$ even to a dummy
- and there is a simple program C, the checker, that verifies the validity of the proof.

Four Examples

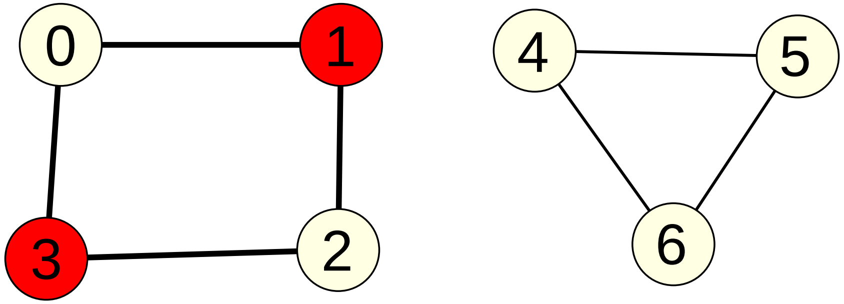
Testing Bipartiteness

Maximum Matchings

Planarity Testing

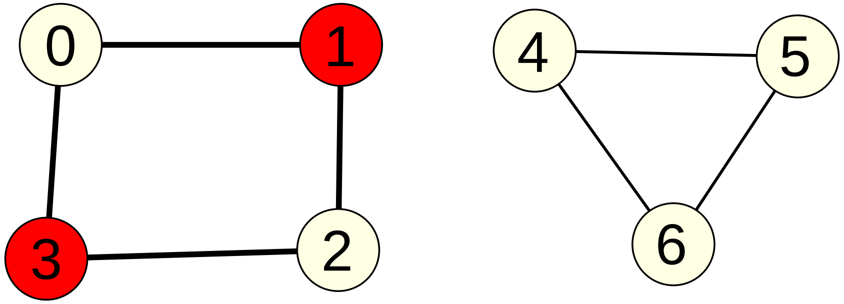
Convex Hulls

Example I: Bipartite Graphs



- Is a given graph G bipartite?
- Two-coloring witnesses bipartiteness
- Odd cycle witnesses non-bipartiteness

Example I: Bipartite Graphs

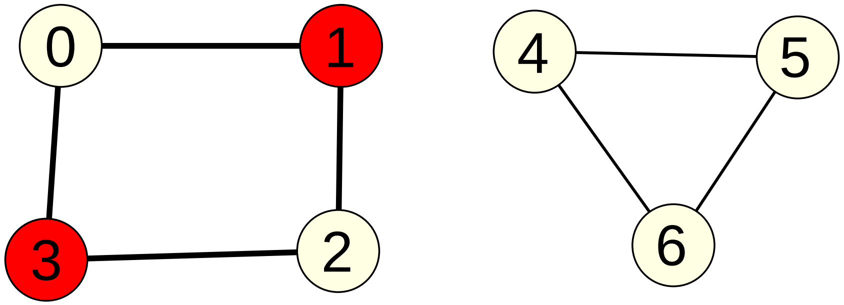


- Is a given graph G bipartite?
- Two-coloring witnesses bipartiteness
- Odd cycle witnesses non-bipartiteness

An Algorithm

- construct a spanning tree of G
- use it to color the vertices with colors red and blue
- check for all non-tree edges: do endpoints have distinct colors?
- if yes, the graph is bipartite and the coloring proves it
- if no, declare the graph non-bipartite:

Example I: Bipartite Graphs



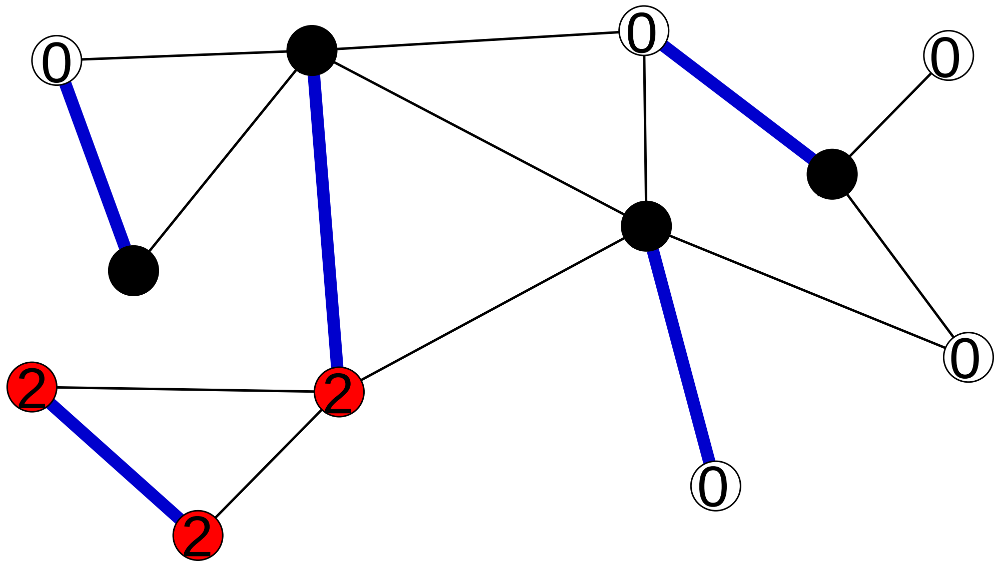
- Is a given graph G bipartite?
- Two-coloring witnesses bipartiteness
- Odd cycle witnesses non-bipartiteness

An Algorithm

- construct a spanning tree of G
- use it to color the vertices with colors red and blue
- check for all non-tree edges: do endpoints have distinct colors?
- if yes, the graph is bipartite and the coloring proves it
- if no, declare the graph non-bipartite: Let $e = \{u, v\}$ be a non-tree edge with equal colored endpoints
  - e together with the tree path from u to v is an odd cycle
  - tree path has even length since u and v have the same color

Example II: Maximum Matchings

- A matching M is a set of edges no two of which share an endpoint

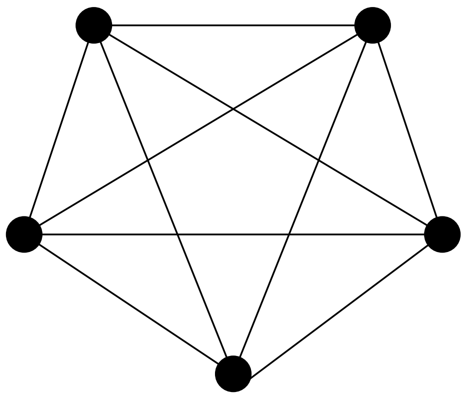


- The coloring certifies that M is of maximum cardinality:
  - Each edge have either two red or at least one black endpoint.
  - Therefore, any matching can use at most one edge with two red endpoints and at most four edges with a black endpoint.
  - The matching shown attains the lower bound.

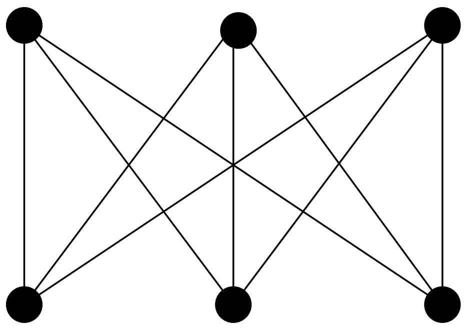
Example III: Planarity Testing

- 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

LEDAbook, Chapter 9



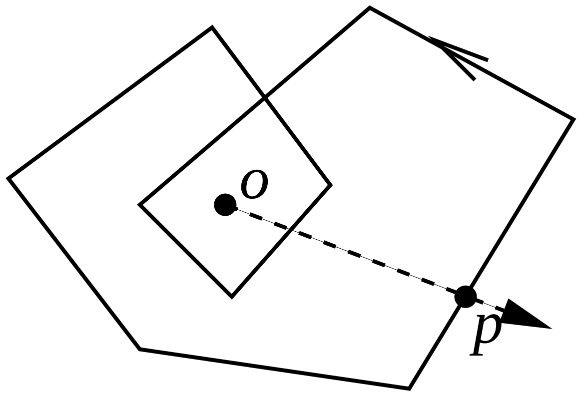
K_5



$K_{3,3}$

Example IV: Convex Hulls

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

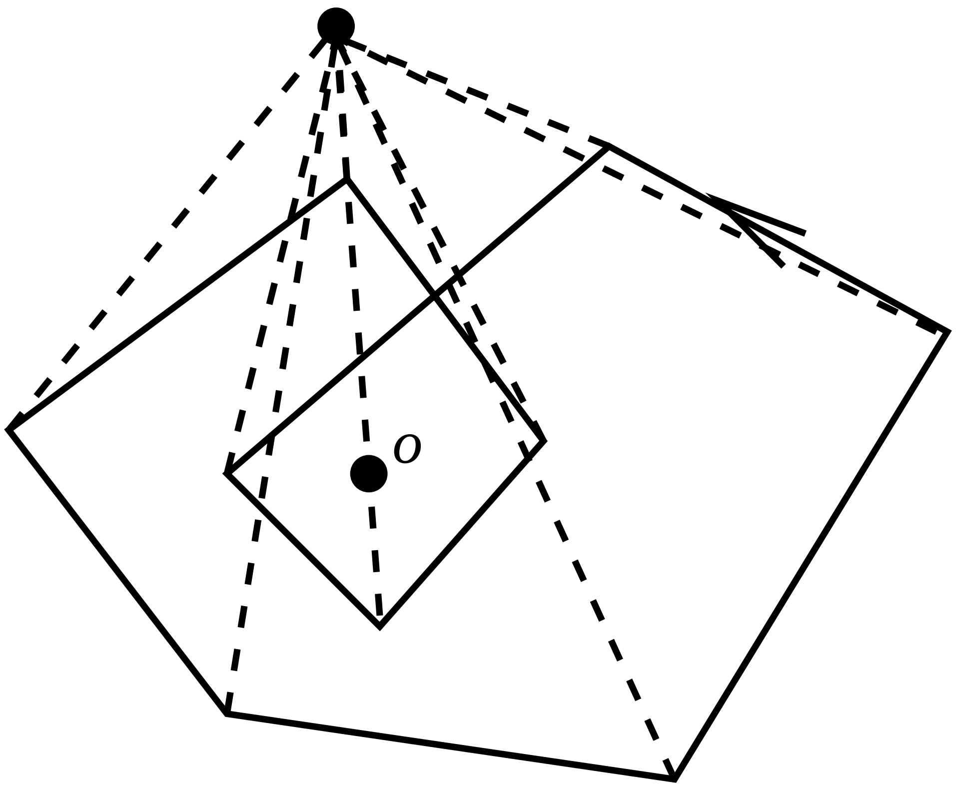


FACT: F is convex iff it passes the following three tests [MNSSSS]

1. check local convexity at every ridge
2. $o =$ center of gravity of all vertices
   check whether o is on the negative side of all facets
3. $p =$ center of gravity of vertices of some facet f
   check whether ray $\overrightarrow{op}$ 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 algorithms 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 witness property and checkers is feasible

Odds and Ends

- General techniques
  - Linear programming duality
  - Characterization theorems
  - Program composition
- Probabilistic programs and checkers
- Reactive Systems (data structures)
- History: an ancient concept
  - al-Kwarizmi: multiplication   Euclid: gcd
  - primal-dual algorithms in combinatorial optimization
  - Blum et al.: Programs that check their work
  - Mehlhorn and Näher make it design principle for LEDA
  - Kratsch/McConnell/Mehlhorn/Spinrad (SODA 2003) coin name
  - McConnell/M/Näher,Schweitzer (2010): 80 page survey

The Message

- Certifying algorithms are much superior to non-certifying algorithms.
- For complex algorithmic tasks only certifying algorithms are satisfactory.
- Consequence: A change of how algorithms are taught, researched and used.

Current Projects

- Universality
- Formal verification
- 3-connectivity of graphs

Universality

MAX-PLANCK-GESELLSCHAFT

- Does every problem have a certifying algorithm? Can every program be converted into a certifying one?

Universality

- Does every problem have a certifying algorithm? Can every program be converted into a certifying one?

- I know 50+ certifying algorithms, see survey by McConnell/M/Näher/Schweitzer (to appear in CS Review),

Universality

- Does every problem have a certifying algorithm? Can every program be converted into a certifying one?
- I know 50+ certifying algorithms, see survey by McConnell/M/Näher/Schweitzer (to appear in CS Review),
- many programs in LEDA are certifying,

Universality

- Does every problem have a certifying algorithm? Can every program be converted into a certifying one?

- I know 50+ certifying algorithms, see survey by McConnell/M/Näher/Schweitzer (to appear in CS Review),

- many programs in LEDA are certifying, and

- Thm: Every deterministic program can be made certifying without asymptotic loss of efficiency

(at least in principle)

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:
   - $\forall x, y \quad (\exists w\, W(x,y,w)) \quad \text{iff} \quad (y = f(x))$.
   - Given x, y, and w, it is trivial to decide if $W(x,y,w)$ holds
   - $W(x,y,w) \implies (y = f(x))$ has a trivial proof
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

Does every Function have a Certifying Alg?

- Formalize the notion of a certifying algorithm
- 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.

Verification: Why Formal Proofs

- why a formal proof for something that has been proved already?
- standard theorems can only be trusted if a fair number of people have checked the proof, taught the result, . . .
- formal proofs are correct and complete (no hidden assumptions)
- are machine-checked (by a fairly simple program)
- add another layer of trust
- user has to understand even less; all that is needed is trust in the proof checker
- allow to build large libraries of trusted algorithms

Verification I: Witness Property

- bipartite matching: a node cover C is a set of vertices such that every edge has an endpoint in C.

  Let M be a matching and C be a node cover. If $|M| = |C|$, then M has maximum cardinality.

  - map $e \in M$ to its endpoint in C.
  - mapping is well-defined, since C is a node cover
  - mapping is injective, since M is a matching
  - thus $|M| \leq |C|$

- we have formalized the proof in Isabelle (a proof support system)

Verification II: The Checker

- Input for Checker: a graph $G = (V, E)$, a subset M of the edges, a node cover C.
  - check $M \subseteq E$
  - check M is a matching
  - check $C \subseteq V$
  - check C is a node cover
  - check $|M| = |C|$.
- we have written a C-program for the above and verified it in VCC (a verification system for C-programs)
- the checker in LEDA checks only items 2, 4, and 5.

Triconnectivity

- $C \subseteq V$ is a *cut set* if $G \setminus C$ is not connected

  Cut sets of size one, two, three: separation vertex, separation pair, separation triple

- Triconnected graph = a graph with no separation pair.

- Linear Time Decision Algorithms: Hopcroft/Tarjan (73) and Miller/Ramachandran (92)

  algs return separation pair or state that graph is triconnected

  Gutwenger/Mutzel (00): former alg misclassifies some non-triconnected graphs, provide a correction

Contractible Edges

- Contraction of an edge xy: contract x and y into a single vertex and remove parallel edges and self-loops
- If $mindeg(G) \geq 3$ and G is not triconnected, then G/xy is not triconnected

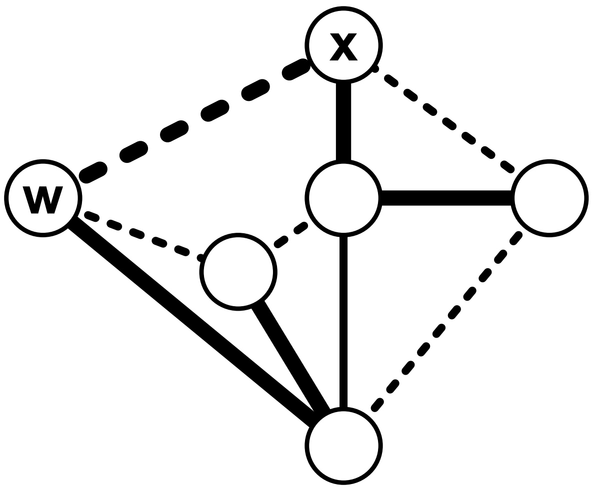
  If $G = G_n, G_{n-1}, \ldots, G_4 = K_4$, $mindeg(G_i) \geq 3$ and G_{i-1} is obtained from G_i by a contraction, then G is triconnected.
- Tutte (61): Every triconnected graph contains a *contractible* edge, i.e, an edge xy such that G/xy is triconnected.
- A certifying algorithm for triconnectivity returns a separation pair if input graph is not triconnected

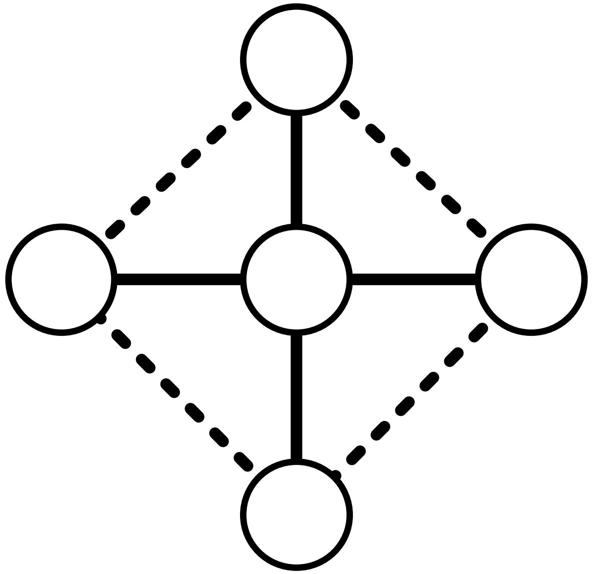
  returns a contraction sequence if input graph is triconnected
- Open Problem: Is there a linear time certifying algorithm for triconnectivity?

$O(n^2)$ is known; Jens M. Schmidt, STACS 2010

Structural Results for Triconnected Graphs

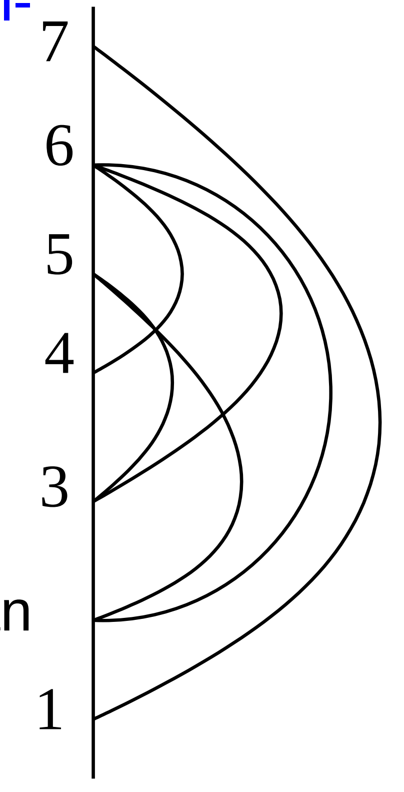
- Tutte: at least one contractible edge
- Ando et al: $\Omega(n)$ contractible edges
- Elmasry/M/Schmidt (2010)
  - every DFS tree contains at least one contractible edge
  - there are DFS trees with exactly one contractible edge
  - there are spanning trees with no contractible edge





Algorithmic Result

Linear-time Certifying Algorithm for Hamiltonian graphs (Elmasry/M/Schmidt (2010))



Hamiltonian Graph = Path + Chords

Why Hamiltonian Graphs:

- HT–algorithm is recursive; merge step is essentially triconnectedness of Hamiltonian graphs.
- MR–alg is based on ear decomposition; in Hamiltonian graph, ears are edges

Technique: DFS-tree is a path; dynamic data structure for testing whether a tree edge is contractible

The Algorithm

- Data structure: $O(1)$ test whether a tree edge is contractible;
- Tree edges are labelled “non-contractible” or “don’t know”
- The algorithm
  - label all tree edges as don’t know;
  - while graph has more than 4 vertices
    - select a tree edge labelled don’t know and test it
    - if contractible, contract and set label of two above and below to don’t know

      else label non-contractible
- $5 \cdot$ # of edges + # of edges labelled “don’t know” decreases in every iteration

Open Problems

- Arrangements of Algebraic Curves
  - numerical algs, symbolic algs, geometric algs
- 3-connectivity of graphs (recently solved by J. Schmidt)
- Formal proofs
- Boolean operations on polygons and polyhedra
- . . .

Summary

- Certifying algs have many advantages over standard algs:
  - can be tested on every input
  - are 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.

When you design your next algorithm,

make it certifying.