



Certifying Algorithms

Kurt Mehlhorn

MPI für Informatik

Saarbrücken

Germany

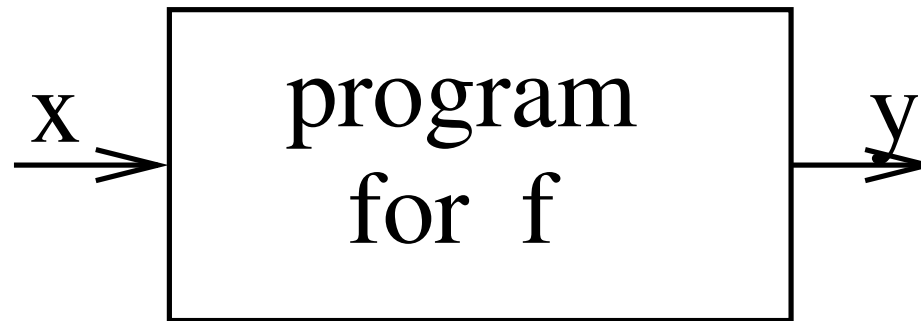
Background



MAX-PLANCK-GESELLSCHAFT

- born in Ingolstadt, Bavaria, 1949, married, three children (29, 27, 24)
- PhD in CS, Cornell University, 1974
- professor CS in Saarbrücken, 1975 - 1990
- director at MPI Computer Science, Saarbrücken, 1991 –
- research interests: data structures, combinatorial and geometric algorithms, combinatorial optimization, algorithm engineering, software library design and implementation
- software libraries
 - LEDA, library of efficient data types and algorithms
 - CGAL, computational geometry algorithms library
 - EXACUS, exact computation with curves and surfaces
 - in all three cases, I participated in the design, in the case of LEDA, I wrote 10% of the code, in the case of EXACUS, I wrote the first code.

The Problem Statement

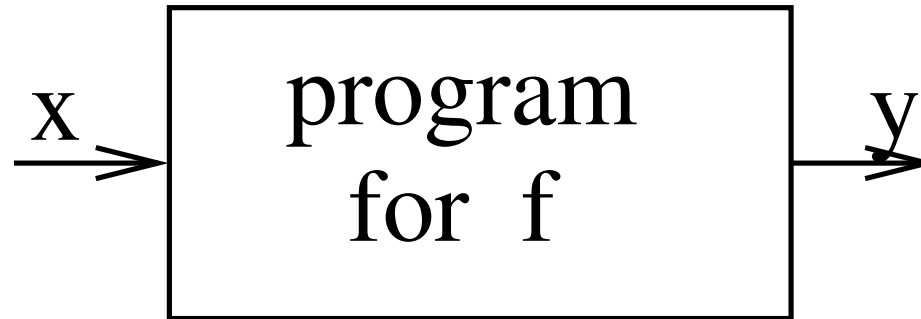


- we want to compute a function $f : X \rightarrow Y$; a user feeds x to the program, the program returns y .
- how can the user be sure that, indeed, $y = f(x)$.
- the user has no way to know, he must trust the program and its author, he is at complete mercy of the program
- I do not like to depend on software in this way, not even for programs written by myself. I know how often I have erred.

The Problem Statement



MAX-PLANCK-GESELLSCHAFT



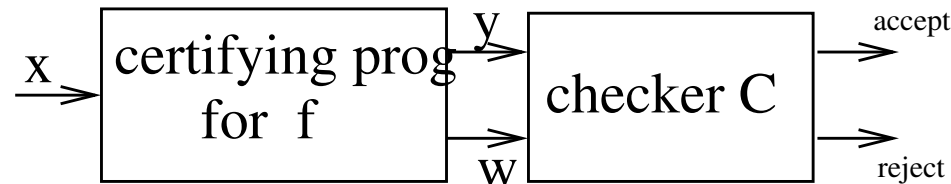
- we want to compute a function $f : X \rightarrow Y$; a user feeds x to the program, the program returns y .
- how can the user be sure that, indeed, $y = f(x)$.
- the user has no way to know, he must trust the program and its author, he is at complete mercy of the program
- I do not like to depend on software in this way, not even for programs written by myself. I know how often I have erred.

programs should justify (prove) their answers in a way that is easily checked by the user of the program.

Certifying Algorithms



MAX-PLANCK-GESELLSCHAFT



- a certifying program returns
 - the function value y and a certificate (witness) w .
- w certifies (proves) the equality $y = f(x)$
- and there is a correct program C , the *checker*, that verifies the validity of the proof.
- properties of formal proofs
 - it is easily checked whether w is a proof of a statement φ
 - if w is a proof of φ , then φ is actually true
- formalization in second half of talk, name introduced in Kratsch, McConnell, Mehlhorn, Spinrad: SODA 2003, related work: Blum et al.: Programs that check their work

Outline of Talk



- problem definition and certifying algorithms
- examples of certifying algorithms
 - testing bipartiteness
 - matchings in graphs
 - planarity testing
 - convex hulls
 - dictionaries and priority queues
 - linear programming
- advantages of certifying algorithms
- programs with non-trivial preconditions
- do certifying algorithms always exist?
- how to design a certifying algorithm?
- verification of checkers
- collaboration of checking and verification

Bipartite Graphs: A Certifying Algorithm



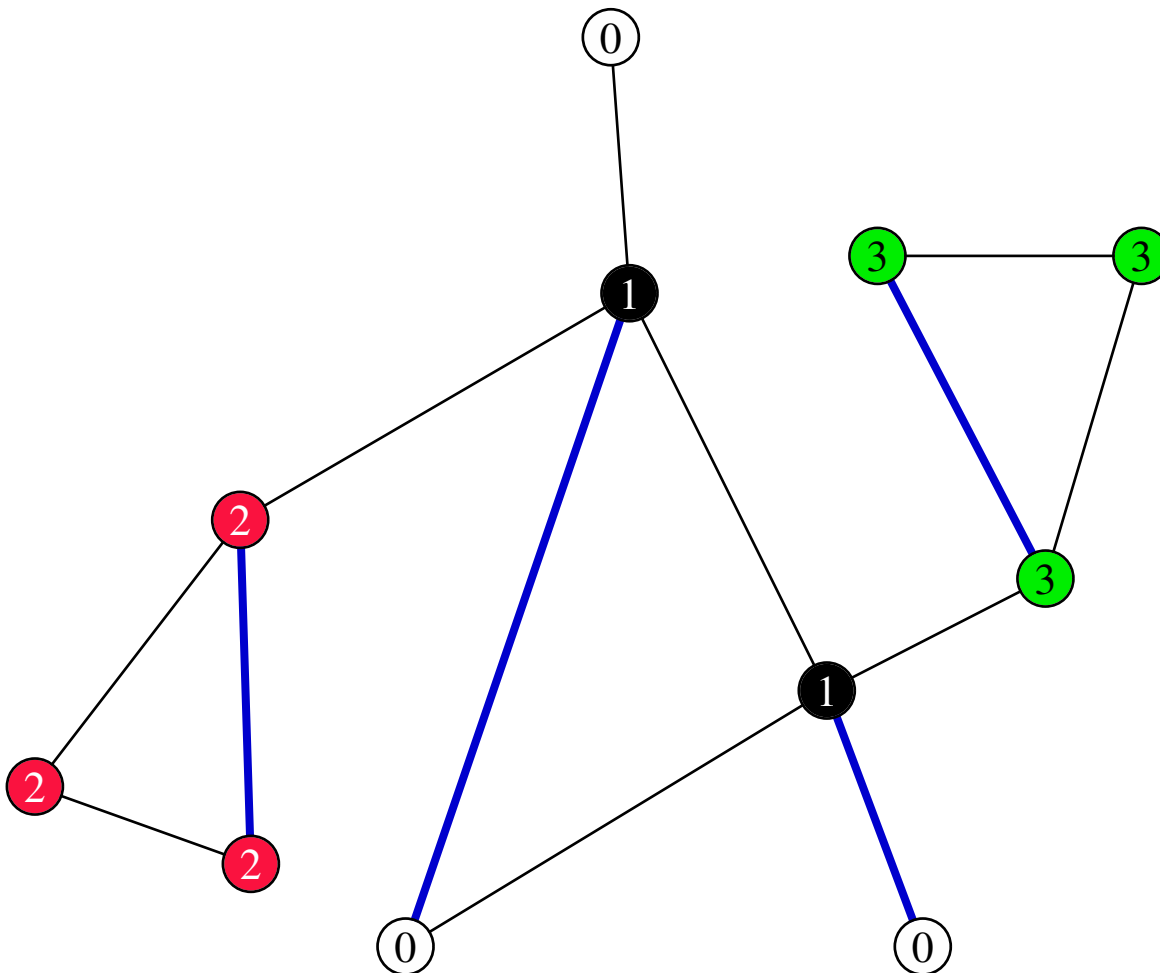
MAX-PLANCK-GESellschaft

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

Matchings



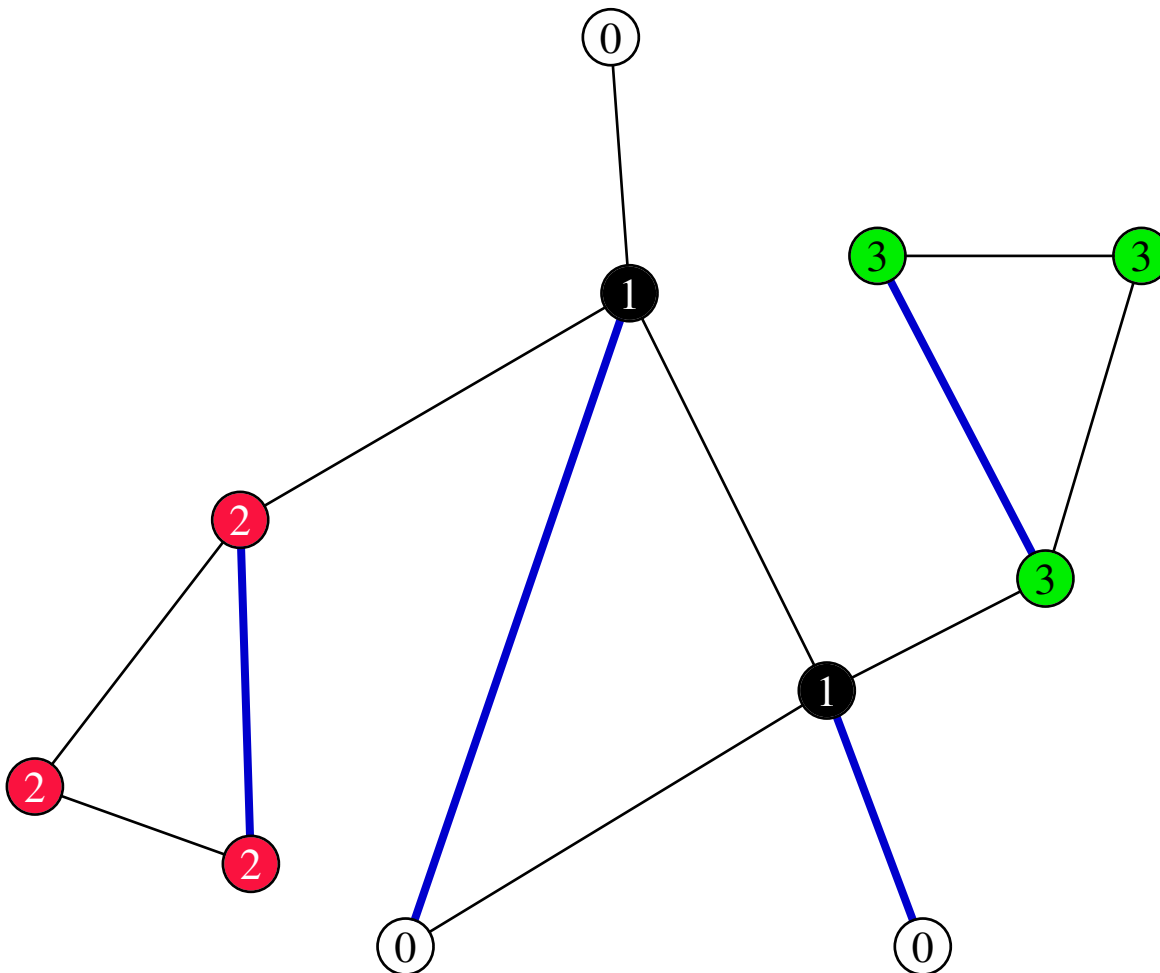
- given a graph, compute a maximum matching
- a matching M is a set of edges no two of which share an endpoint
- a conventional algorithm outputs a maximal matching



Matchings



- given a graph, compute a maximum matching
- a matching M is a set of edges no two of which share an endpoint
- a conventional algorithm outputs a maximal matching



The node labels prove optimality.

Every edge is either incident to a node labeled 1 or connects two nodes labeled 2 or connects two nodes labeled 3.

Thus no matching can have more than $2 + \lfloor 3/2 \rfloor + \lfloor 3/2 \rfloor = 4$ edges.

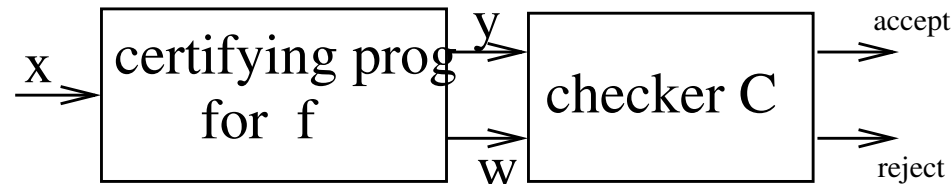
The matching shown has four edges and is hence optimal.

Matchings II



- *odd-set cover* = integer labeling of the nodes of G such that
- every edge of G is either incident to a node labeled 1 or connects two nodes labeled with the same i and $i \geq 2$
- **Lemma:** n_i = number of nodes labeled i . M a matching. If
$$|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor.$$
then M is optimal.
- Let N be any matching. For $i \geq 2$, N_i = edges in N that connect two nodes labeled i , N_1 = remaining edges in N .
- then $|N_i| \leq \lfloor n_i/2 \rfloor$ for $i \geq 2$ and $|N_1| \leq n_1$.
- a *certifying algorithms for maximum cardinality matching* returns a matching M and an odd-set cover OSC satisfying the equality above.
- it is *not* necessary to understand why odd-set covers proving optimality exist. It is only required to understand the Lemma above. Also, a correct program which checks the properties of a matching, an odd-set cover, and the equality above is easy to write.

Certifying Algorithms



- a certifying program returns
 - the function value y and a certificate (witness) w .
- w certifies (proves) the equality $y = f(x)$
- and there is a correct program C , the *checker*, that verifies the validity of the proof.
- in matching example: $y =$ an optimal matching M , $w =$ an odd-set cover with $|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$.
 - it is easily checked whether M is a matching and w is an odd-set cover and $|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$ holds.
 - if the above holds, M is an optimal matching

Advantages: Instance Correctness



MAX-PLANCK-GESELLSCHAFT

- If C accepts (x, y, w) , w is a valid proof of “ $y = f(x)$ ” and hence y is equal to $f(x)$.
- We are certain that our (certifying) program worked correctly on instance x .
- We do not know that the program will work correctly for all inputs, we only know it for instance x .
- If the checker rejects (x, y, w) , w is not a valid certificate for “ $y = f(x)$ ” and we know that our program erred either in the computation of y or the computation of the certificate w .

Advantages: Testing on all Inputs



- **we can test our program on every input x**
 - run the program on x
 - run the triple (x, y, w) through C
 - if C accepts, be happy
- **not only on inputs for which we already know the right answer by other means.**
- the latter is a major short-coming of testing non-certifying programs

Advantages: Confinement of Error



MAX-PLANCK-GESELLSCHAFT

- Whenever a certifying program fails on an input, the checker catches the error.
- In a system consisting of certifying components, errors are caught by the failing module and do not spread to other modules.
- This greatly simplifies the task of finding the source of the error.
- Program modules are usually incorrect or incomplete during much of their development phase.
- Confinement of error is particularly important during this phase.
- When I program a certifying algorithm, I implement the checker first.

Trust with Minimal Intellectual Investment: Which intellectual investment does a user have to make in order to trust a certifying program? Not much. First, he has to understand, what constitutes a valid certificate for the equality “ $y = f(x)$ ”, and second he has to verify that the checker is correct.

Verified Checkers: As we will see in the examples in later sections, checkers are frequently so simple, that their formal verification is feasible. Also, they are frequently so efficient compared to the program itself, that we may be willing to write them in a less efficient programming language which eases verification.
Program in C++, checker in ML.

Black-Box Programs: In order to develop trust in a certifying program there is no need to have access to the source text of the program. It suffices to have access to the source of the checker since only its correctness needs to be verified. In other words, the intellectual property can be kept secret and yet trust can be developed.

Advantages VI



MAX-PLANCK-GESELLSCHAFT

Safer Programming: Turning a correct algorithm into a correct program is an error-prone task. An algorithm is the description of a problem solving method intended for human consumption. It is usually described in natural language, pseudo-code, mathematical notation or a mixture thereof. A program is the description of a problem solving method intended for execution by a machine. It is written in a programming language and usually much more detailed than the algorithm. Certifying algorithms are easier to implement correctly than non-certifying algorithms because they can be tested on all inputs.

Hidden Assumptions: A checker can only be written if the problem at hand is rigorously defined. We noticed that some of our specifications in LEDA contained hidden assumptions which were revealed during the design of the checker. For example, an early version of our biconnected components algorithm assumed that the graph contains no isolated nodes.

Certifying Algorithms: Practical Experience



MAX-PLANCK-GESELLSCHAFT

- developed the concept of certifying algorithms during the design of the LEDA
- started to build LEDA in 1989: we implemented conventional algorithms and some checks for assertions and post-conditions and we tested extensively.
- Nevertheless at least two major programs were incorrect when we first releases them: the planarity test and the convex hull algorithms in arbitrary dimensional spaces
- In the attempt to correct the errors in these programs we developed the concept of certifying program and reported about it in ICALP97.
- For the LEDA book, many algorithms were reimplemented and a large number of the algorithms in LEDA were made certifying, in particular, the graph and geometry algorithms.
- However, there are still large parts which are non-certifying, e.g., all algorithms working on the number types of LEDA.

The Need for Certifying Programs: Warning Examples

- LEDA 2.0 planarity test was incorrect
- Rhino3d (a CAD systems) fails to compute correct intersection of two cylinders 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}}
```

Linear Programming



$$\text{maximize } c^T x \quad \text{subject to } Ax \leq b \quad x \geq 0$$

- linear programming is a most powerful algorithmic paradigm
- there is no linear programming solver that is guaranteed to solve large-scale linear programs to optimality. Every existing solver may return suboptimal or infeasible solutions.

Problem				CPLEX				Exact Verification
Name	C	R	NZ	T	V	Res	RelObjErr	T
degen3	1504	1818	26230	8.08	0	opt	6.91e-16	8.79
etamacro	401	688	2489	0.13	10	dfeas	1.50e-16	1.11
fffff800	525	854	6235	0.09	0	opt	0.00e+00	4.41
pilot.we	737	2789	9218	3.8	0	opt	2.93e-11	1654.64
scsd6	148	1350	5666	0.1	13	dfeas	0.00e+00	0.52

Dhiflaoui/Funke/Kwappik/M/Seel/Schömer/Schulte/Weber: SODA 03

open problem: exact solution of very large sparse linear systems

Certifying Algorithms, a Challenge for Algorithmics



MAX-PLANCK-GESELLSCHAFT

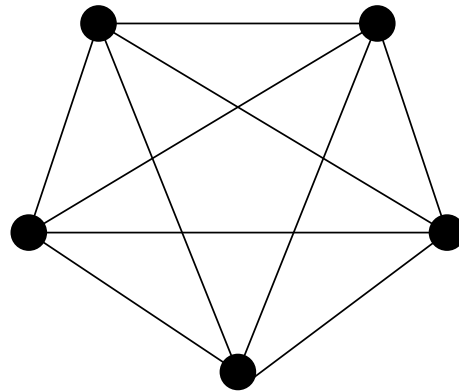
- most existing algorithms are non-certifying.
- it is a challenge for algorithmics to find certifying algorithms which are as efficient as the existing non-certifying ones.

Planarity Testing

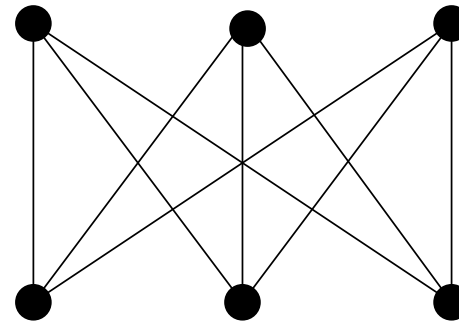


- given a graph G , decide whether it is planar
- Tarjan (76): planarity can be tested in linear time
- a story and a demo
- combinatorial planar embedding is a witness for planarity
- Chiba et al (85): planar embedding of a planar G in linear time
- Kuratowski subgraph is a witness for non-planarity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

LEDAbook, Chapter 9



K_5

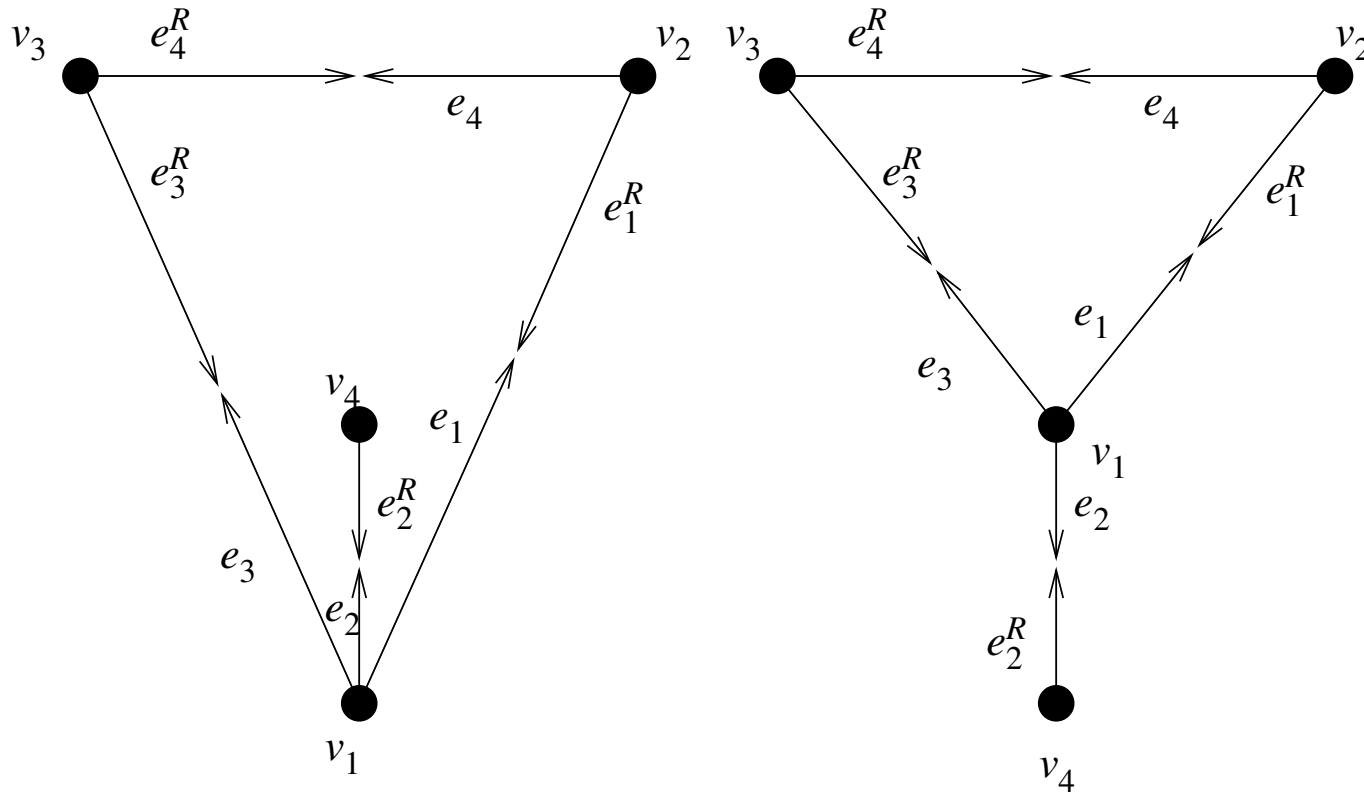


$K_{3,3}$

Planarity Testing: Checking the Witness I



- combinatorial embedding: graph + cyclic order on the edges incident to any vertex



- combinatorial planar embedding: combinatorial embedding such that there is a plane drawing conforming to the cyclic orderings
- it is easily checked whether a combinatorial embedding is planar

Euler's Formula



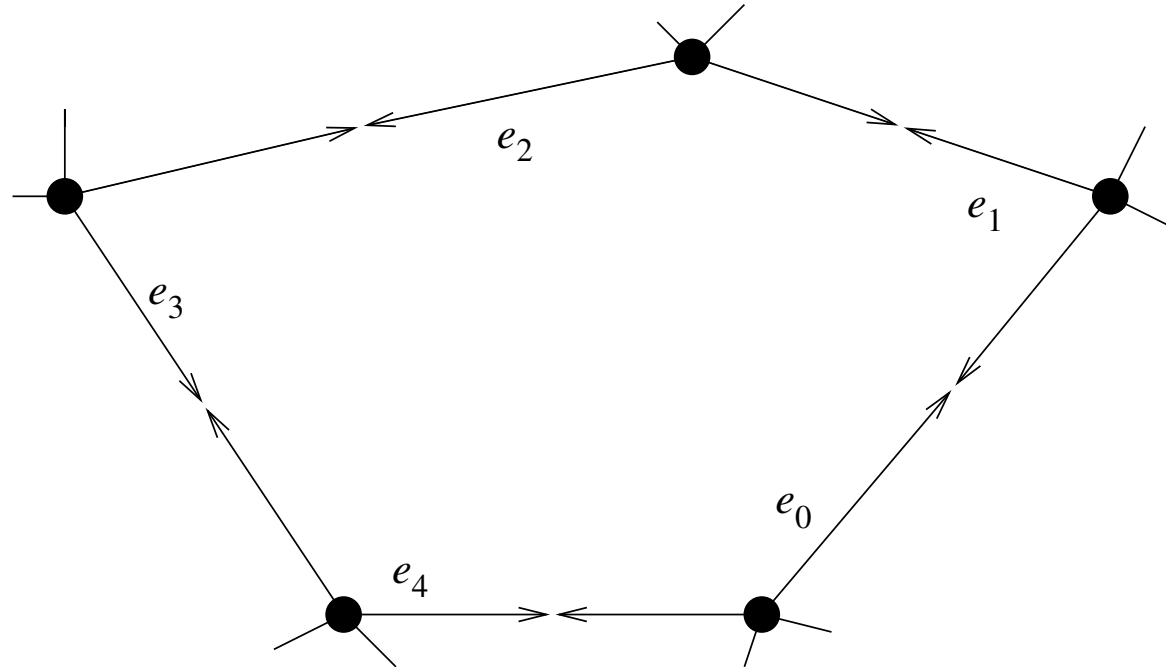
- consider a planar drawing of a connected graph: n vertices, e edges, f regions
- **Fact:** $f - e + n = 2$
- an easy induction on e : a tree has $f = 1$, $e = n - 1$
- thus $f - e + n = 1 - (n - 1) + n = 2$.
- adding an edge splits a region and thus increments f and e
- thus $f - e + n$ does not change.

Planarity Testing: Checking the Witness II



MAX-PLANCK-GESELLSCHAFT

- face cycles



- face cycles are defined for combinatorial embeddings.
- **Theorem[Euler, Poincaré]** A combinatorial embedding of a connected graph is a combinatorial planar embedding iff

$$f - e + n = 2$$

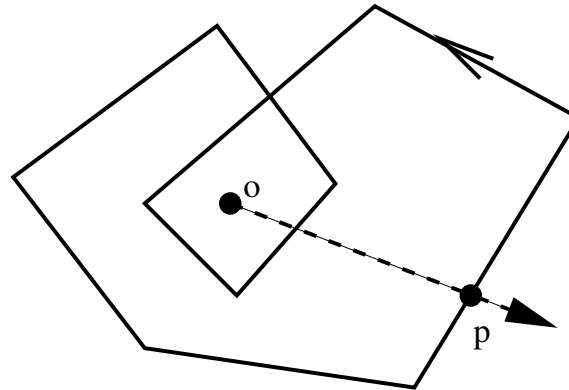
- I proved one direction on the previous slide
- theorem = easy check whether a combinatorial embedding is planar.

Convex Hulls



MAX-PLANCK-GESELLSCHAFT

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. $0 =$ center of gravity of all vertices
check whether 0 is on the negative side of all facets
3. $p =$ center of gravity of vertices of some facet f
check whether ray $\vec{0p}$ intersects closure of facet different from f

Discussion I



- one ray in third test
- test is fast (linear in size of F) and simple
 1. = scalar product
 2. = sign of linear function
 3. = linear system solving
 - Let f' be any facet and let h' be a supporting hyperplane.
 - $q = r \cap h'$ is a point.
 - q is a convex combination of vertices of f' and this combination is **unique** (since f' is a simplex). Thus, solve

$$q = \sum_{j=1}^d \lambda_j \cdot v_j$$

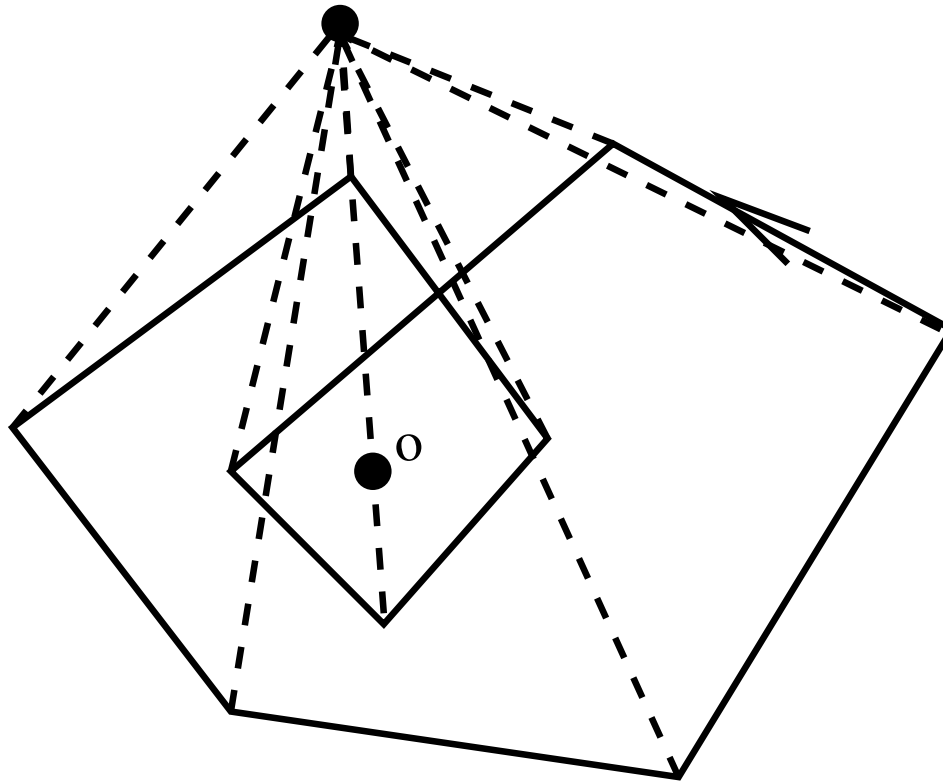
and check whether $0 \leq \lambda_j \leq 1$ for all j .

Sufficiency of Test is a Non-Trivial Claim



MAX-PLANCK-GESELLSCHAFT

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



Proof of Sufficiency



MAX-PLANCK-GESELLSCHAFT

Lemma: All rays which intersect only the interiors of facets intersect the same number of facets.

Vary $r = \vec{0p}$ but keep p on interiors of facets and ridges. When r crosses a ridge, number of intersections either stays the same or changes by ± 2 .

Latter case is excluded.

View situation in plane orthogonal to ridge.

In our situation: any such ray intersects the interior of exactly one facet (since the ray through the center of our chosen facet does).

Proof of Sufficiency, Continued



MAX-PLANCK-GESELLSCHAFT

no ray starting in o intersects the interior of more than one facet.

For any facet f of F , let h_f be the supporting hyperplane. Let A be the arrangement of these hyperplanes and let K be the cell of A containing o . K is convex. We show that F is the boundary of K .

Consider any facet g of K . We claim that it is contained in a facet of F . Assume otherwise. We have $g \subseteq h_f$ for some facet f of F . **But**, there is a ridge r separating f from g in h_f . View situation in plane orthogonal to ridge and passing through o . Let f' be the other facet of F incident to r . Observe, that g must be on the negative side of f' . Thus

and hence $bd K$ equal to F .

Certification of Convex Hulls



MAX-PLANCK-GESELLSCHAFT

Task: Given a set S of points and a hyper-surface F verify that F is the boundary of the convex hull of S .

We know already how to check the convexity of F . So let us assume that F has passed the convexity test and let P be the convex polytope whose boundary is F . It suffices to verify that

- every vertex of P is a point in S and that
- every point of S is contained in P .

The first item is fairly easy to check. If the vertices of P are specified as pointers to elements in S the check is trivial. If the vertices in S are specified by their coordinate tuples the check involves a dictionary lookup.

The second condition is much harder to check. A simple method would be to check every point of S against every facet of F . But this takes time $O(n \cdot \# \text{ of facets})$. The best (randomized) construction algs take only $O(\# \text{ of facets})$.

A Certifying Alg for Convex Hulls



MAX-PLANCK-GESELLSCHAFT

- give a set S of points in R^d
- use randomized incremental construction to construct a triangulation of its convex hull
 - a point p outside current hull adds a simplex $S(F, p)$ for every hull facet F visible to it
 - a point p inside the current hull is stored with the simplex containing it.
- data structures witnesses the correctness of the construction
 - check convexity of boundary as described on preceding slides
 - check that all vertices of triangulation belong to S
 - check that every point in S is stored with some simplex of the triangulation

Five Coloring Planar Graphs



MAX-PLANCK-GESELLSCHAFT

- Task: color the vertices of a planar graph such that the endpoints of every edge have distinct colors.
- **Fact:** every simple (= no parallel edges) planar graph has a vertex of degree at most five
- let v be a vertex of degree at most five:
 - if v does not exist, declare G non-planar. Witness: too many edges
 - if v has degree four or less, solve $G - v$ recursively. Color v .
 - if v has degree five, there must be neighbors u and w which are not connected by an edge. Otherwise, the neighbors form a K_5 and G is non-planar. K_5 is witness.
 - delete v and merge u and w . This maintains planarity. Color the simplified graph. Unmerge u and w and use the free color for v .
- algorithm either five-colors G or proves that G is non-planar.