Trustworthy Graph Algorithms

MFCS 2019

Mohammad Abdulaziz[1] Tobias Nipkow[1]
Kurt Mehlhorn

max planck institut
informatik

SIC Saarland
Informatics Campus

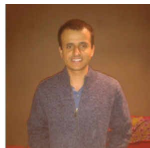
[1] Technical University Munich

Overview

This is a difficult talk to give. Usually, ... The talk is based on a collaboration with two experts in interactive theorem proving.

At MFCS '89, I announced that Stefan Näher and I would build LEDA, a library of efficient data types and algorithms. The library would be encompassing, easy-to-use, efficient and correct.

Tobias Nipkow



Mohammad Abdulaziz

Overview

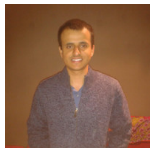
This is a difficult talk to give. Usually, ... The talk is based on a collaboration with two experts in interactive theorem proving.

At MFCS ’89, I announced that Stefan Näher and I would build LEDA, a library of efficient data types and algorithms. The library would be encompassing, easy-to-use, efficient and correct.

However, some of our implementations were **incorrect**, e.g., planarity test. Since then I am also very much interested in correct implementations.

- LEDA (MFCS 1989)
- Certifying Algorithms (MFCS 1998)
- Formal Verification of Checkers
- Formal Verification of Complex Algorithms (MFCS 2019)
- Future

Tobias Nipkow



Mohammad Abdulaziz

Library of Efficient Data Structures and Algorithms

joint work with S. Näher and C. Uhrig

- When I asked former students, ...

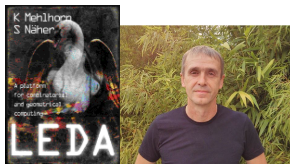
Library of Efficient Data Structures and Algorithms

joint work with S. Näher and C. Uhrig

- When I asked former students, ...
- Insight (1988): Writing books and articles does not suffice.

Must turn knowledge of the field into software that is

  - easy-to-use,
  - correct, and
  - efficient.



- Started LEDA project in 1988, later CGAL, STXXL, SCIL
- In use at thousands of academic and industrial sites

Algorithmic Solutions GmbH

Dijkstra's Algorithm in LEDA

algorithm + LEDA = program

```
template <class NT>
void DIJKSTRA_T(const graph& G, node s,const edge_array<NT>& c,
                node_array<NT>& dist, node_array<edge>& pred)
{ node_pq<NT>  PQ(G);
  node v; edge e;
  dist[s] = 0; PQ.insert(s,0);
  forall_nodes(v,G) pred[v] = nil;
  while (!PQ.empty())
  { node u = PQ.del_min();
    NT du = dist[u];
    forall_adj_edges(e,u)
    { v = G.opposite(u,e);
      NT dv = du + c[e];
      if (pred[v] == nil && v != s )
        PQ.insert(v,c);
      else if (dv < dist[v]) PQ.decrease_p(v,dv);
           else continue;
      dist[v] = dv; pred[v] = e;
    }
  }
}
```

LEDA: Guarantees and Promises

Mathematical Guarantees

- Algorithms are correct.

Promises, Engineering Guarantees

- Combinatorial and geometric algorithms can be formulated in a natural way: Algorithm + LEDA = Program
- As a consequence, our users will be more effective and will find it easier to write efficient and correct code.
- Our implementations are correct.

LEDA: Guarantees and Promises

Mathematical Guarantees

- Algorithms are correct.

Promises, Engineering Guarantees

- Combinatorial and geometric algorithms can be formulated in a natural way: Algorithm + LEDA = Program
- As a consequence, our users will be more effective and will find it easier to write efficient and correct code.
- Our implementations are correct.

But,

Some of our programs were incorrect.

But,

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

- We had followed the state of the art, however,

But,

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

- We had followed the state of the art, however,
- there was no scientific basis available for geometric computations, and

We and others created a basis over the next 20 years.

But,

Some of our programs were incorrect.

What had gone wrong and what did we do about it?

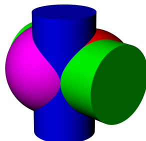
- We had followed the state of the art, however,
- there was no scientific basis available for geometric computations, and

  We and others created a basis over the next 20 years.
- we made mistakes and wrote incorrect programs.

  Adopted new design principle: certifying algorithms.

Not only our Programs are Incorrect

- LEDA 2.0 planarity test was incorrect
- Rhino3d (a CAD systems) fails to compute correct intersection of two cyclinders and two spheres

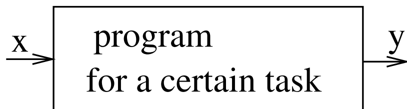


- CPLEX (a linear programming solver) fails on benchmark problem *etamacro*.
- Mathematica 4.2 (a mathematics systems) fails to solve a small integer linear program

```
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 Problem



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

  y is indeed the correct output for input x?

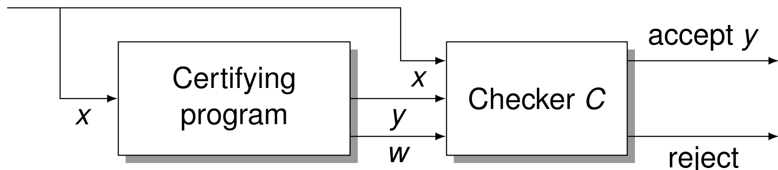
  The user has no way to know.

analogy: construction company

The Proposal (in 95)

Programs must justify (prove) their answers in a way that is easily checked by their users.

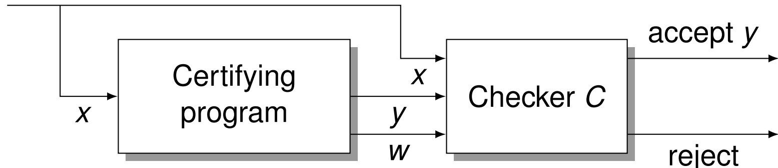
Certifying Algorithms



On input x, a certifying algorithm computes

- the function value y and
- a witness w. (convincing evidence that y is the correct output for x)

Certifying Algorithms

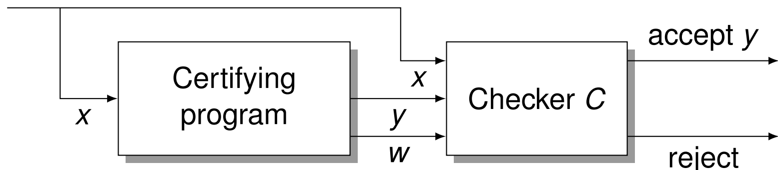


On input x, a certifying algorithm computes

- the function value y and
- a witness w. (convincing evidence that y is the correct output for x)

w is inspected by either the human user of the certifying program,

Certifying Algorithms



On input x, a certifying algorithm computes

- the function value y and
- a witness w. (convincing evidence that y is the correct output for x)

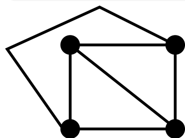
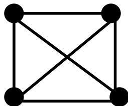
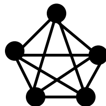
w is inspected by either the human user of the certifying program,

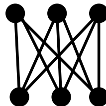
or more elegantly, by a checker program C.

Example: Planarity Test

Planar Graph

A graph is planar if it can be drawn in the plane without edge crossings.



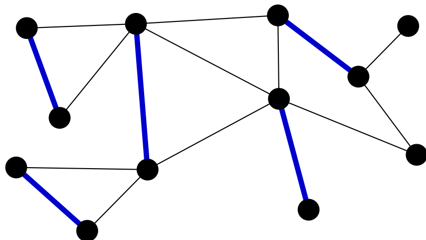
planar drawing planar graph nonplanar graphs

Fact: Every non-planar graph contains a Kuratowski graph.

Story and Demo

Example: Maximum Cardinality Matchings

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



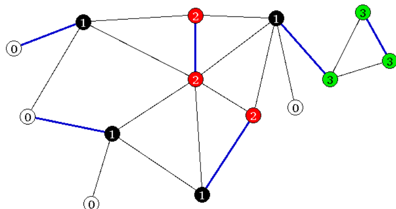
- The blue edges form a matching of maximum cardinality; this is non-obvious as two vertices are unmatched.
- A conventional algorithm outputs the set of blue edges.

Maximum Cardinality Matching: A Certifying Alg

Edmonds' Lemma: Let M be a matching in a graph G and let ℓ be a labelling of the vertices with non-negative integers such that for each edge $e = (u, v)$ either $\ell(u) = \ell(v) \geq 2$ or $1 \in \{\ell(u), \ell(v)\}$. Then

$$|M| \leq n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor ,$$

where n_i is the number of vertices labelled i. In particular, if equality holds, M is max-card-matching.



- $n_1 = 4$, $n_2 = 3$, $n_3 = 3$.
- no matching has more than $4 + \lfloor 3/2 \rfloor + \lfloor 3/2 \rfloor = 6$ edges.
- $|M| = 6$

Maximum Cardinality Matching: A Certifying Alg

Edmonds' Lemma: Let M be a matching in a graph G and let ℓ be a labelling of the vertices with non-negative integers such that for each edge $e = (u, v)$ either $\ell(u) = \ell(v) \geq 2$ or $1 \in \{\ell(u), \ell(v)\}$. Then

$$|M| \leq n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor ,$$

where n_i is the number of vertices labelled i. In particular, if equality holds, M is max-card-matching.

- Let M_1 be the edges in M having at least one endpoint labelled 1 and, for $i \geq 2$, let M_i be the edges in M having both endpoints labelled i.
- $M = M_1 \cup M_2 \cup M_3 \cup \dots$
- $|M_1| \leq n_1$ and $|M_i| \leq n_i/2$ for $i \geq 2$.

Maximum Cardinality Matching: A Certifying Alg

Edmonds' Lemma: Let M be a matching in a graph G and let ℓ be a labelling of the vertices with non-negative integers such that for each edge $e = (u, v)$ either $\ell(u) = \ell(v) \geq 2$ or $1 \in \{\ell(u), \ell(v)\}$. Then

$$|M| \leq n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor ,$$

where n_i is the number of vertices labelled i. In particular, if equality holds, M is max-card-matching.

A certifying algorithm for max-card matching on input G outputs M and ℓ such that

- M is a matching in G,
- ℓ is a legal node labelling, and
- $|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$.

Maximum Cardinality Matching: A Certifying Alg

Edmonds' Lemma: Let M be a matching in a graph G and let ℓ be a labelling of the vertices with non-negative integers such that for each edge $e = (u, v)$ either $\ell(u) = \ell(v) \geq 2$ or $1 \in \{\ell(u), \ell(v)\}$. Then

$$|M| \leq n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor ,$$

where n_i is the number of vertices labelled i. In particular, if equality holds, M is max-card-matching.

The labelling ℓ is the witness.

Witness property: If M is a matching, ℓ is a legal labelling, and $|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$ then M has maximum cardinality.

Existence of a Witness: If M has maximum cardinality, there is a witness. This is the hard direction of Edmonds' Theorem.

The Checker Program for Maximum Cardinality Matching

```
bool CHECK_MAX_CARD_MATCHING(const graph& G,const list<edge>& M,
                             const node_array<int>& OSC)
{ int n = Max(2,G.number_of_nodes());
  array<int> count(n); for (int i = 0; i < n; i++) count[i] = 0;
  node v; edge e;

  forall_nodes(v,G)
  { if ( OSC[v] < 0 || OSC[v] >= n ) return_false("illegal label");
    count[OSC[v]]++;
  }

  forall_edges(e,G)
  { node v = G.source(e); node w = G.target(e);
    if ( v == w || OSC[v] == 1 || OSC[w] == 1 ||
            ( OSC[v] == OSC[w] && OSC[v] >= 2) ) continue;
    return_false("OSC is not a cover");
  }

  node_array<int> deg_in_M(G,0);
  forall(e,M) {deg_in_M(G.source(e))++; deg_in_M(G.target(e)); }
  forall_nodes(v,G) if (deg_in_M(v) > 1) return_false("M is not a matching");

  int S = count[1];
  for (int i = 2; i <= n; i++) S += count[i]/2;
  if ( S != M.length() ) return_false("OSC does not prove optimality");

  return true;
}
```

Certifying Algorithms: Guarantees and Promises

Mathematical Guarantees

- If x, y, w satisfy $\varphi(x)$ and the witness predicate $\mathcal{W}(x, y, w)$, then the postcondition $\psi(x, y)$ holds.
- If x satisfies $\varphi(x)$, the algorithm computes a y and w such that $\mathcal{W}(x, y, w)$.

Promises, Engineering Guarantees

- Checker programs are very simple and hence correct.
- If the checker accepts the witness, the output is correct. If the output is incorrect, the checker will reject it.
- The programs have been executed on many inputs and the checkers never fired.
- You can trust our programs without understanding them.

Certifying Algorithms: Guarantees and Promises

Mathematical Guarantees

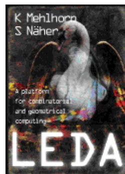
- If x, y, w satisfy $\varphi(x)$ and the witness predicate $\mathcal{W}(x, y, w)$, then the postcondition $\psi(x, y)$ holds.
- If x satisfies $\varphi(x)$, the algorithm computes a y and w such that $\mathcal{W}(x, y, w)$.

Promises, Engineering Guarantees

- Checker programs are very simple and hence correct.
- If the checker accepts the witness, the output is correct. If the output is incorrect, the checker will reject it.
- The programs have been executed on many inputs and the checkers never fired.
- I would accept a 10.000 Euro correctness bet . . .

History

- I do not claim that we invented the concept; it is rather an old concept
  - al-Kwarizmi (780 – 850): multiplication
  - extended Euclid ($\approx$ 1700): gcd
  - primal-dual algorithms in combinatorial optimization
  - Blum et al.: Programs that check their work
- I do claim that Stefan Näher and I were the first (1995) to adopt the concept as the design principle for a software project: By now, almost all algs in LEDA are certifying.



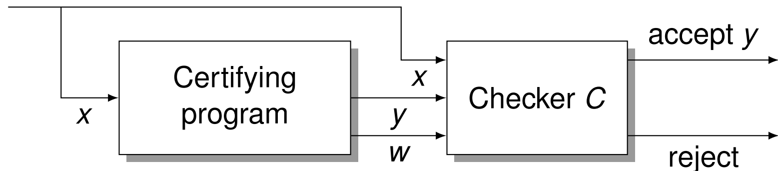
- McConnell/M/Näher/Schweitzer (2010): 80 page survey

Summary of Indoctrination Part

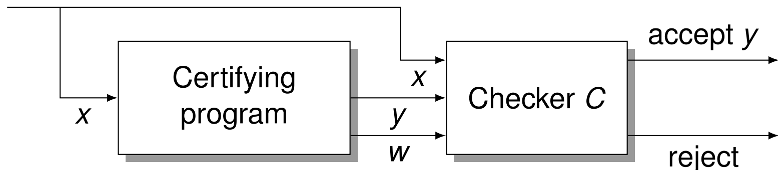
When you design your next algorithm, make it certifying.

When you implement your next algorithm, also implement a checker.

Who Checks the Checker?



Who Checks the Checker?



Answer till 2011: Checkers are simple programs, and hence, their correctness is not an issue.

Answer since 2011: Checkers are simple programs, and hence, we can prove their correctness using formal mathematics.

Formal Mathematics

- Mathematics is carried out in a formal language.
- Proofs are machine-checked.
- Only correct statements can be proven!!!
- We use Isabelle/HOL (L. Paulson, T. Nipkow)
  - Interactive theorem prover
  - Proofs are machine-checked; a small kernel must be trusted.
  - Automatic reasoning tools (term rewriting engine, tableaux prover, decision procedures). They do not have to be trusted, but suggest proofs that are then checked.
  - has been used to prove Gödel incompleteness, substantial analysis results (probabilities, Green's theorem & ODE solvers), correctness of an operating system kernel, ...
  - Archive of formal proofs: 490 articles, $>$ 10,000 Lemmata

Square Root of Two is not a Rational

```
|    1| theory Sqrt2
|    2| imports Main
|    3| begin
|    4| theorem sqrt2_not_rational:
|    5|   "sqrt (real 2) ∉ Q"
|    6| proof
|    7|   let ?x = "sqrt (real 2)"
|    8|   assume "?x ∈ Q"
|    9|   then obtain m n :: nat where
|   10|     sqrt_rat: "?x = real m / real n" and lowest_terms: "coprime m n"
|   11|     by (rule Rats_abs_nat_div_natE)
|   12|   hence "real (m^2) = ?x^2 * real (n^2)" by (auto simp add: power2_eq_square)
|   13|   hence eq: "m^2 = 2 * n^2" using of_nat_eq_iff power2_eq_square by fastforce
|   14|   hence "2 dvd m^2" by simp
|   15|   hence "2 dvd m" by simp
|   16|   have "2 dvd n" proof
|   17|     from 2 dvd m obtain k where "m = 2 * k"
|   18|     with eq have "2 * n^2 = 2^2 * k^2" by simp
|   19|     hence "2 dvd n^2" by simp
|   20|     thus "2 dvd n" by simp
|   21|   qed
|   22|   with 2 dvd m have "2 dvd gcd m n" by (rule gcd_greatest)
|   23|   with lowest_terms have "2 dvd 1" by simp
|   24|   thus False using odd_one by blast
|   25| qed
|   26| end
|   27|
|   28|
```

Verification of the Max-Card Checker

We gave formal proofs for the following theorems:

Lemma: If M is a matching in G and ℓ is a legal node labelling and $|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$, then M is a maximum cardinality matching. LP-duality

Lemma: The checker program always halts.

Lemma: On input G, M and ℓ, the checker program returns true if and only if the hypothesis of the first Lemma holds.

Verification of the Max-Card Checker

We gave formal proofs for the following theorems:

Lemma: If M is a matching in G and ℓ is a legal node labelling and $|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$, then M is a maximum cardinality matching. LP-duality

Lemma: The checker program always halts.

Lemma: On input G, M and ℓ, the checker program returns true if and only if the hypothesis of the first Lemma holds.

The proof attempt for the third Lemma failed at first, because the checker program does not check that M is a subset of the edges of G. After adding

```
forall(e,M) if (G.is_edge(e) == false)
                return_false("M is not a subset of E");
```

the proof went through.

E. Alkassar, S. Böhme, KM, Ch. Rizkallah: Verification of Certifying Computations, CAV '11.
L. Noschinski, Ch. Rizkallah, KM: Verification of certifying computations through . . . , NASA Formal Methods '14.
E. Alkassar, S. Böhme, KM, Ch. Rizkallah: A Framework for the Verification of Certifying Computations, JAR '14.

Formal Verification of Checkers: Guarantees and Promises

Mathematical Guarantees

- Checker correctness and witness property are formally verified.
- If the checker accepts, the output is correct.
- If the output is incorrect, the checker will catch it.

Promises, Engineering Guarantees

- The programs have been executed on many inputs and the checkers never fired.
- You can trust our programs without understanding them.

Caveat: have not verified C++-programs, but C-programs derived from them.

Why Formal Proofs?

- Software plays a crucial role in modern society; software failures can lead to substantial damage (money, lives).
- Correctness of software requires proof; testing cannot establish the absence of errors, only the presence of errors.
- Mathematical theorems are accepted after a complex social process; a penciel-and-paper proof is part of this process, but also talks, discussions, questioning, ...
- The same process does work for algorithm, but it does not work for software, because implementations are very detailed, are meant for machines and not humans, and hence correctness proofs of implementations do not appeal to a large audience. However, a large audience is needed for the social process.

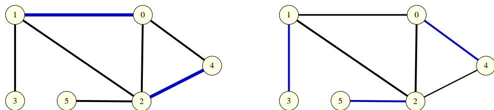
Formal Proofs Will Never Become the Standard

Formal proofs are extremely tedious and hence this approach will never fly. It will stay an obsession of some academics and will never become wide-spread practice.

- Building theories: matching is maximum iff there is no augmenting path; odd-set-cover is dual of matching.
- Liability law: According to liability law in many countries, an engineer/company is liable if he/she/it has not followed the state of the art. So academia only has to push the state of the art. (courtesy of W. Paul)

Verification of Edmond's Blossom-Shrinking Alg

- Alg repeatedly searches for an augmenting path with respect to the current matching. Initially, the current matching is empty. If augmenting path exists, augmentation increases the size of the matching by one. Otherwise, the current matching has maximum cardinality.
- Augmenting path = alternating path connecting two free (= unmatched) vertices.



- Description of alg + correctness proof: 6 pages in LEDAbook, description of implementation: 10 pages.
- Formal correctness proof of alg: 18,000 lines, 250 pages.

More Details of the Search for Augmenting Paths

Grow alternating trees rooted at free vertices.

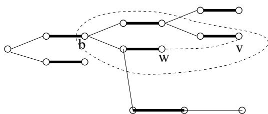
Growth: v even depth and $\{v, w\}$ not considered before.

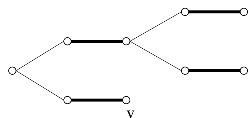
- If w not in a tree yet: add w (at odd level) and its mate (at even level).
- If w already in a tree and odd level: skip
- If w already in a tree and even level:

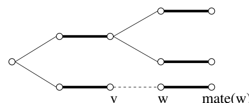
  augmenting path, if w in different tree.

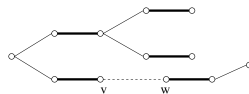
  blossom, if w in same tree. Shrink blossom and continue search in shrunken graph G'.

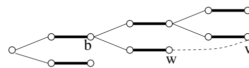
  If augmenting path in G', lift to original graph. Otherwise, no aug-path.

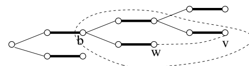












Key Lemmas

Lemma: M is not maximum cardinality iff there is an augmenting path with respect to it.

Lemma: If B is a blossom wrt. M in G, then there is an augmenting path wrt. M in G iff there is an augmenting path wrt. M/B in G/B.

Lemma: The tree building process finds an augmenting path if there is one.

Alternatively:

Lemma: If the tree building process finds neither an augmenting path nor a blossom, then one can construct a node labelling proving optimality.

Lemma: Assume tree building detects a blossom B wrt. M in G. If there is an augm. path wrt. M/B in G/B, then there is one wrt. M in G. If there is none, one can lift the labelling proving optimality of M/B in G/B to a proof of optimality of M in G.

The Verification Process

- Tobias Nipkow spent 6 weeks in SB in the fall of 2018 (mid October to end of November). We started the project and did first formalizations of the alternative approach.
- Mohammad joined the project in January 2019 and worked about 3 months full time on the project. He did all the formalizations following the standard approach.
- Tobias provided hints and suggestions for improvements and elegance.
- I proofread the formalizations of the definitions, lemmas, and theorems (not the proofs). It is easier to prove correct theorems and lemmas than to prove incorrect ones. This is true for paper and pencil proofs and even more true for formal proofs.
- Mohammad found simplications for some of the Lemmas, in particular for one direction of Berge's lemma.

Summary of Second Part

Formal correct proofs of complex combinatorial algorithms are doable.

Summary of Second Part

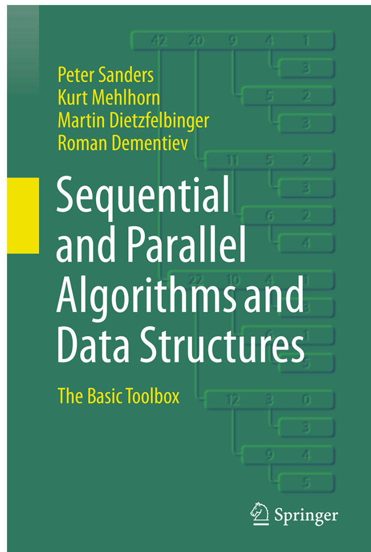
Formal correct proofs of complex combinatorial algorithms are doable.

It still requires experts, but this is changing as the tools are getting more advanced and more students are trained in using them.

There is Still a Lot to Do

- We verified a slow version of the algorithm, not the most sophisticated version of it.
  - Warm start of tree growing process.
  - Fast blossom shrinking with parallel tree-walk and union-find
- We verified an algorithm not an implementation.
- We verified one of the many algorithms/implementations in LEDA
- We abstracted away from the underlying infrastructure: memory management, graph data type, data structures.

The Tool Box



- Will appear in the next days.
- Covers sequential and parallel algorithms and data structures
- Also treats algorithm engineering aspects.
- And, of course, we treat certification.