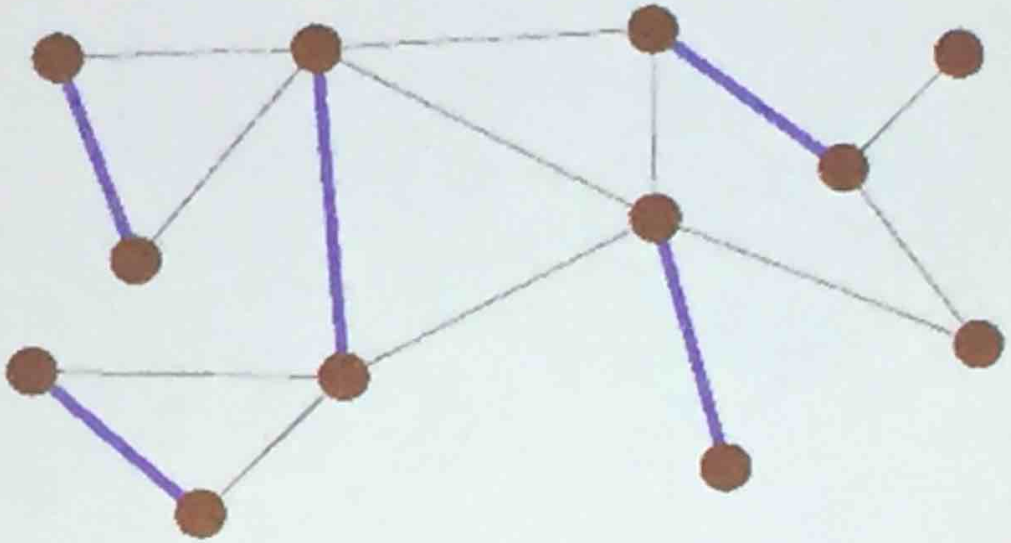
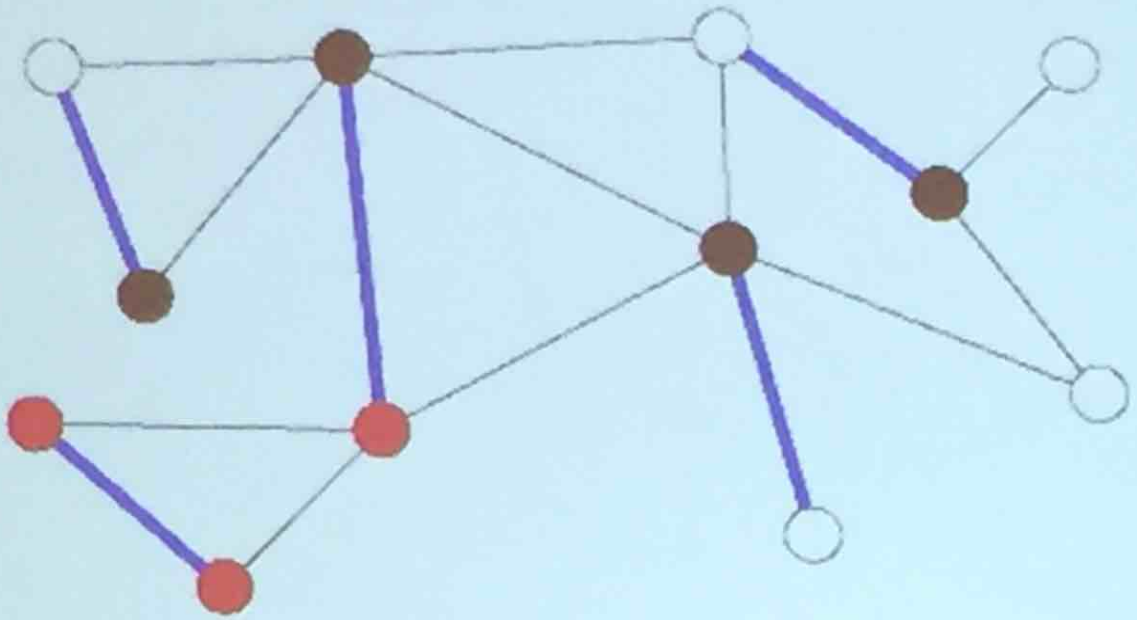
Example II: Maximum Cardinality Matchings

- Nodes (black dots) are customers of a dating service. An edge connecting two customers indicates that they are willing to date. A dating plan is a conflict-free set of dates.



- The blue edges form a dating plan of maximum cardinality; this is non-obvious as two customers are unmatched.
- A conventional algorithm outputs the set of blue edges. Correct, but unsatisfactory.

Maximum Cardinality Matching: A Certifying Alg

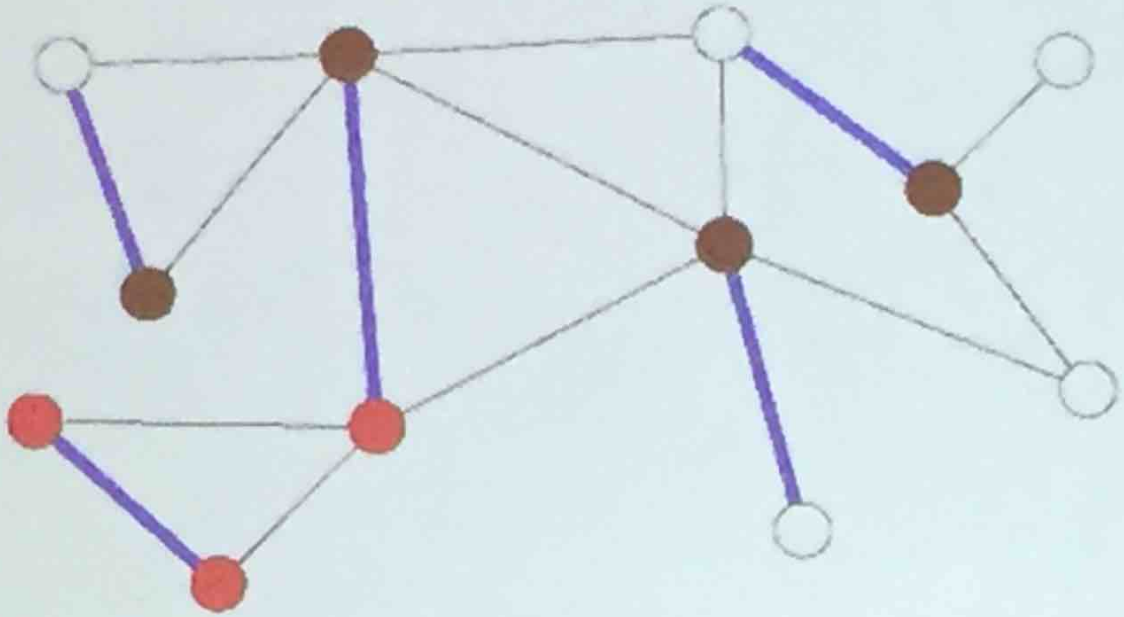


We scheduled five dates.

Why is this optimal?

- Every possible date (edge) involves a black customer or pairs two red customers.
- There are four black customers and hence at most four dates involving a black customer.
- There are three red customers and hence at most one date pairing two red customers.

Maximum Cardinality Matching: A Certifying Alg



We scheduled five dates.

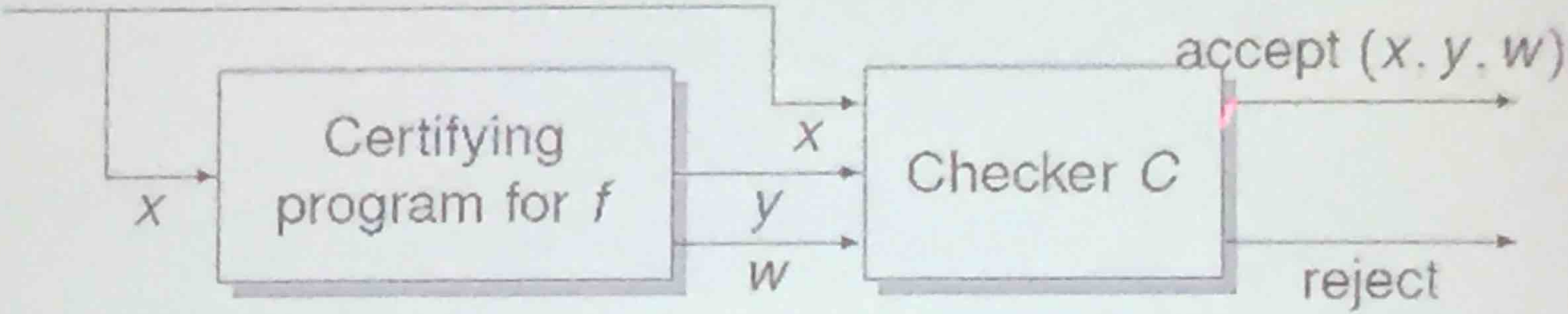
Why is this optimal?

- Every possible date (edge) involves a black customer or pairs two red customers.
- There are four black customers and hence at most four dates involving a black customer.
- There are three red customers and hence at most one date pairing two red customers.
- Thus there cannot be more than five dates.

The Advantages of Certifying Algorithms

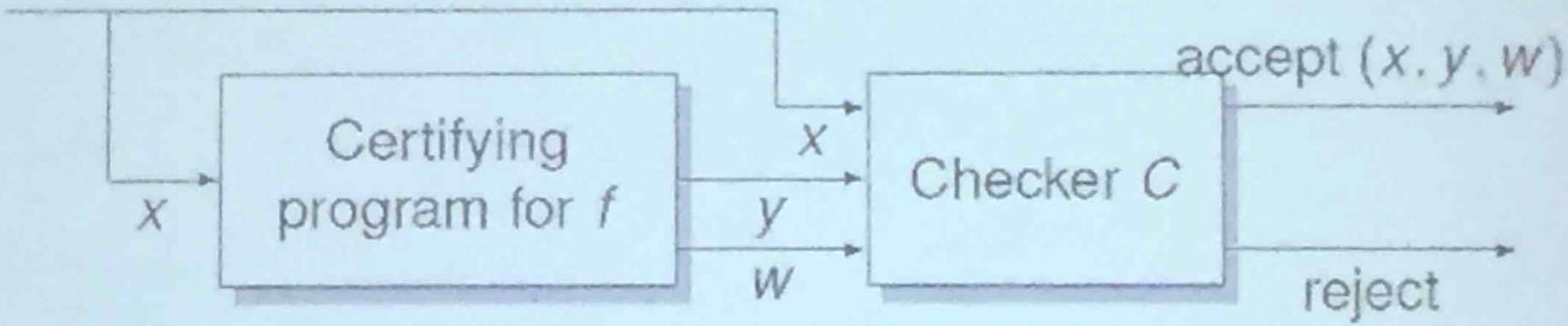
- Certifying algs can be tested on
  - **any** input
  - and not just on inputs for which the result is known.
- Certifying algorithms are dependable:
  - Either give the correct answer
  - or notice that they have erred. $\Rightarrow$ confinement of error
- Computation as a service
  - There is no need 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.
- Algorithms explain their work
  - The witness explains the result of the computation in simple terms.

A Certifying Program for a Function f



- On input x, 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.

A Certifying Program for a Function f



- On input x, 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.

But, why should the checker program be correct?

Who Checks the Checker?

How can we be sure that the checker programs are correct?

My answer up to 2011: Because they are so simple.

Answer now: Because we can prove their correctness in a formal system (Isabelle/HOL). Even higher level of trust.

What do we formally verify?

1. The correctness theorem (appropriate coloring of the customers proves optimality of the schedule)
2. Checker always halts and either rejects or accepts.
3. Checker accepts iff output + witness satisfy the assumptions of correctness theorem.

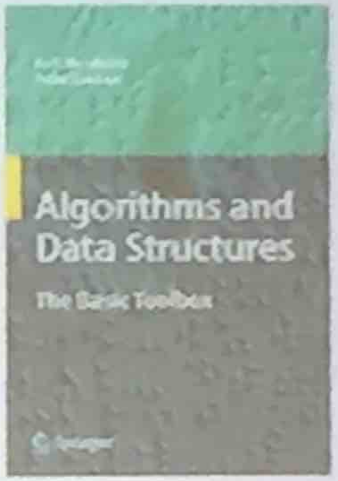
History

- I do not claim that I invented the concept; it is an old concept
  - al-Kwarizmi: multiplication
  - extended Euclid: gcd
  - primal-dual algorithms in combinatorial optimization
  - Blum et al.: Programs that check their work
- I do claim that Näher and I were the first (1995) to adopt the concept as the design principle for a large library project: LEDA

  (Library of Efficient Data Types and Algorithms)
- Kratsch/McConnell/M/Spinrad (SODA 2003) coin name
- McConnell/M/Näher/Schweitzer (2010): 80 page survey
- Alkassar/Böhme/M/Rizkallah (2014): formal verification of checkers

Summary

- **Only certifying algs are good algs**
- Certifying algs have many advantages over standard algs:
  - every run is a test
  - notice when they erred
  - can be relied on without knowing code
  - are a way to computation as a service and to algorithms that explain their work.
- Formal verification of checkers and formal proof of witness property are feasible
- Most programs in the LEDA system are certifying. All algorithms taught in basic algorithm courses can be made certifying.



When you design your next algorithm, make it certifying.