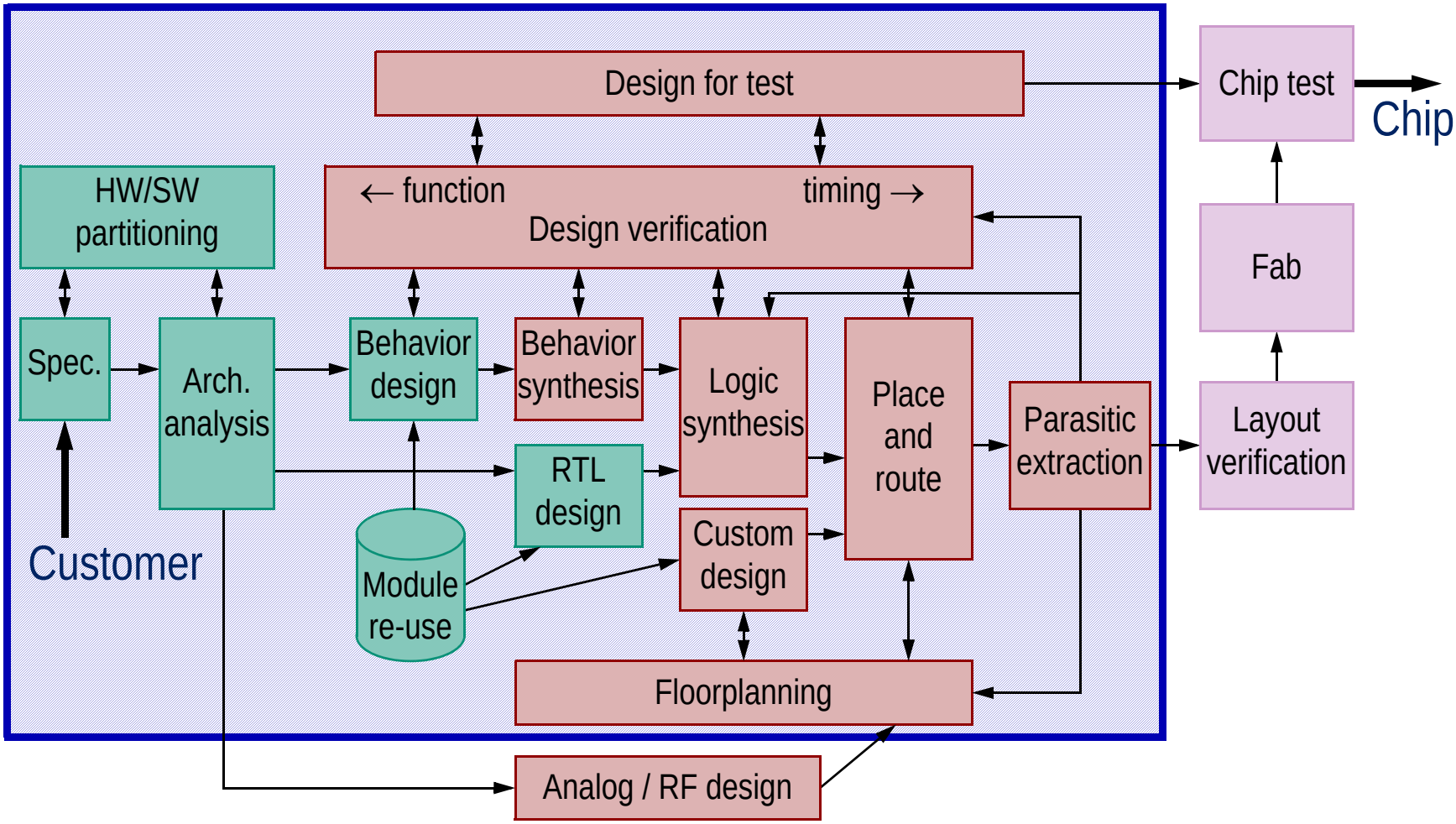
DAT110
Methods for Electronic System Design and Verification

Per Larsson-Edefors
VLSI Research Group

Lecture 9: Discrete Mathematics and Optimization Strategies for EDA.

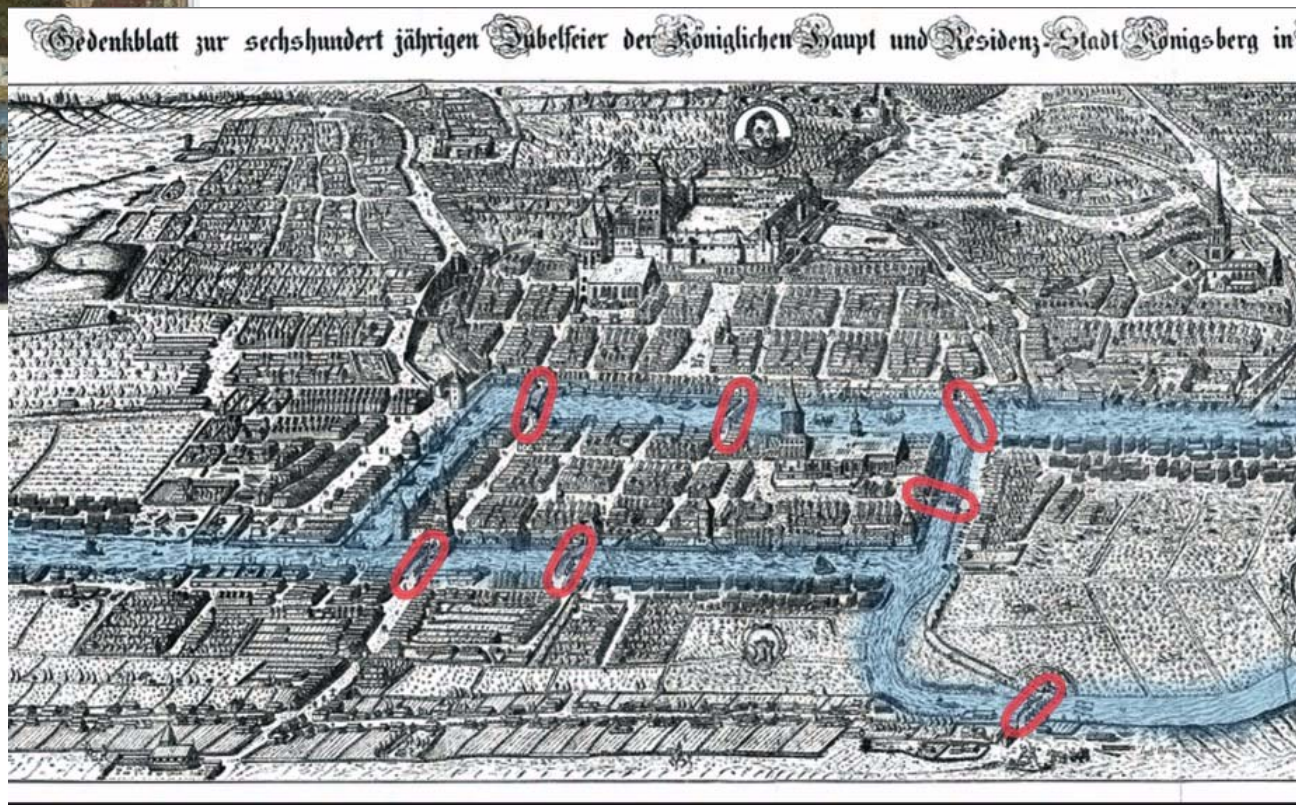
Present Scenario: Discrete Math



Mathematics

- Language, an agreement of scientists and engineers: “never divide something with 0” etc.
- Tool for modeling of our analog world; *f*(*x*, ...): analysis and calculus.
- Tool for solving equations: matrix descriptions and linear algebra.
- Tool for analyzing and optimizing structures/topologies.

The Seven Bridges of Königsberg

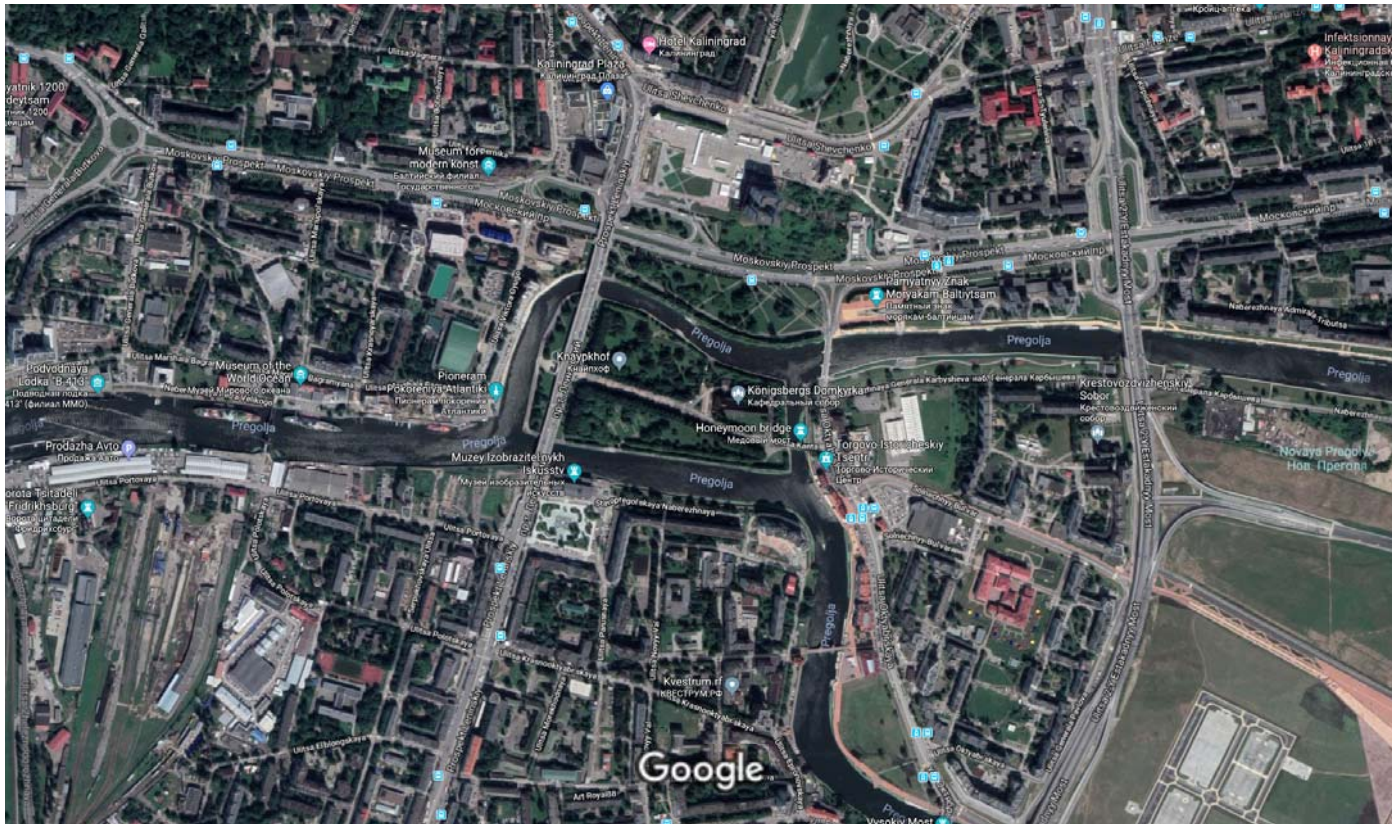


Source: http://www.leonhardeuler.com/

KÖNIGSBERG = KALININGRAD

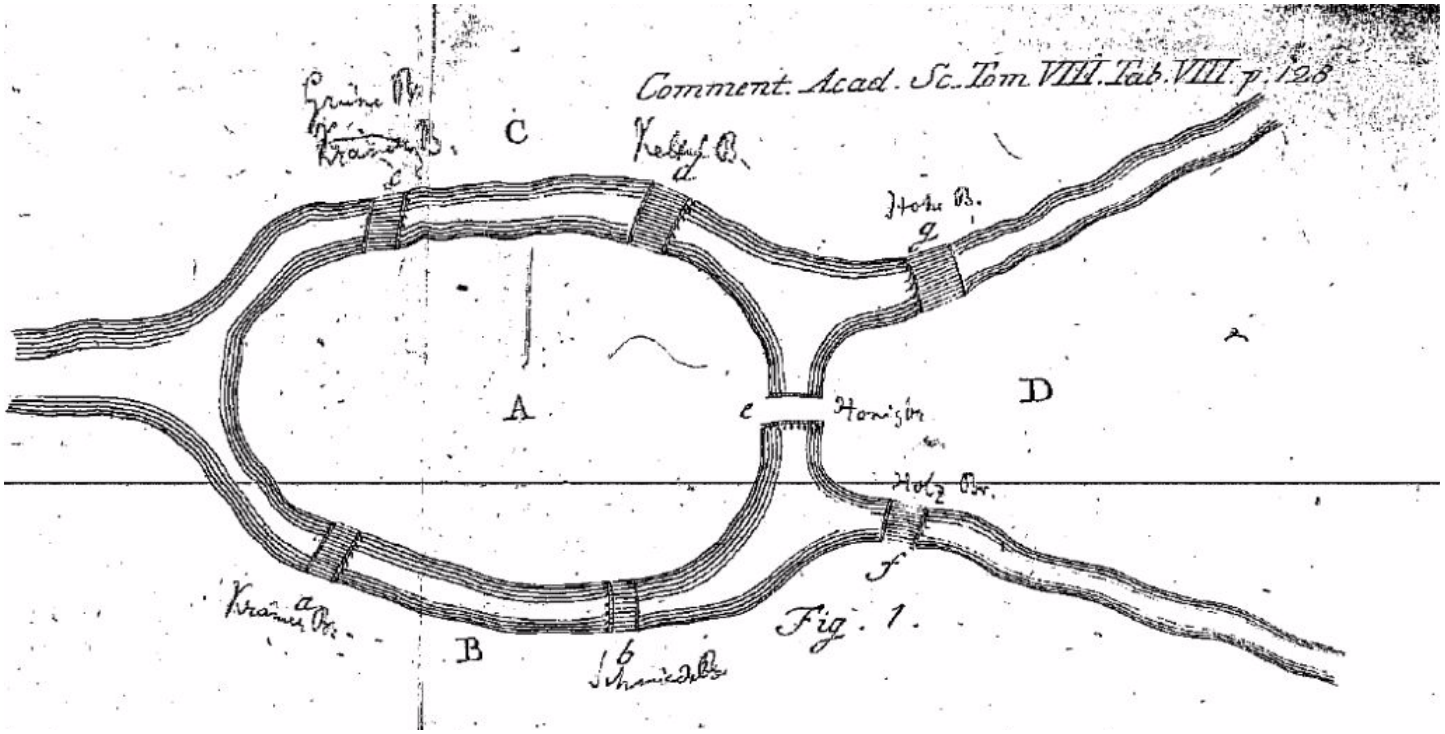
Source: http://www.eatingeuropetours.com/the-most-vegetarian-country-in-europe/

Kaliningrad of Today



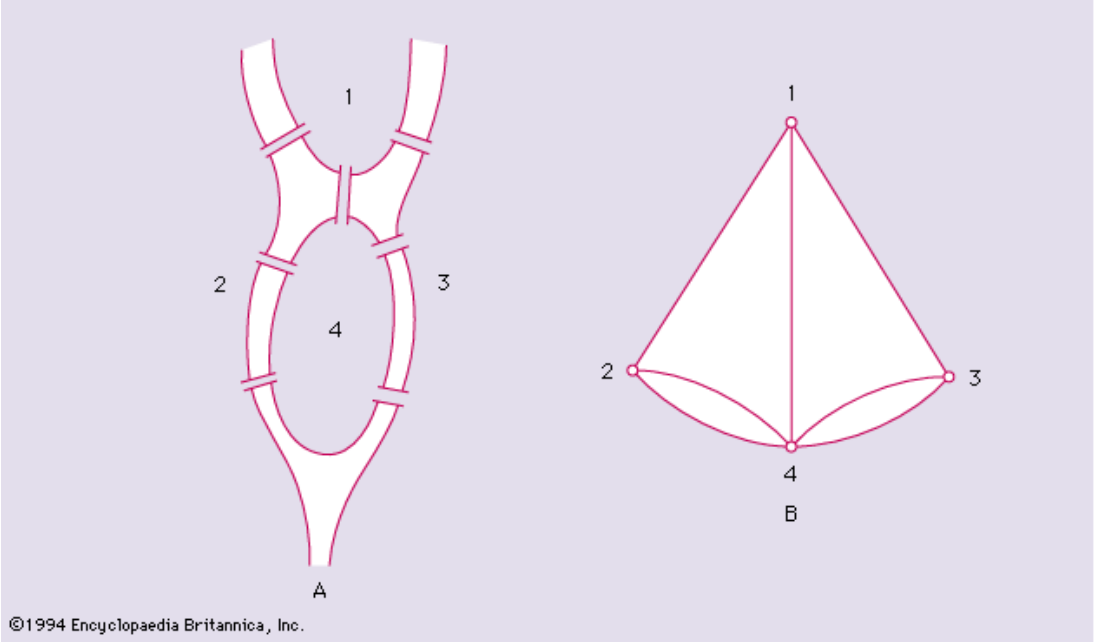
Leonhard Euler's Finding (1735)

- *"If there is a path along edges of a multigraph that traverses each edge once and only once, then there exist at most two vertices of odd degree; furthermore, if the path begins and ends at the same vertex, then no vertices will have odd degree."*



Seven Bridges Representation

- Bridges are edges and land/islands are vertices.
- Edges should be passed once; vertices can be visited many times.
- Three vertices have degree 3, and one has degree 5.

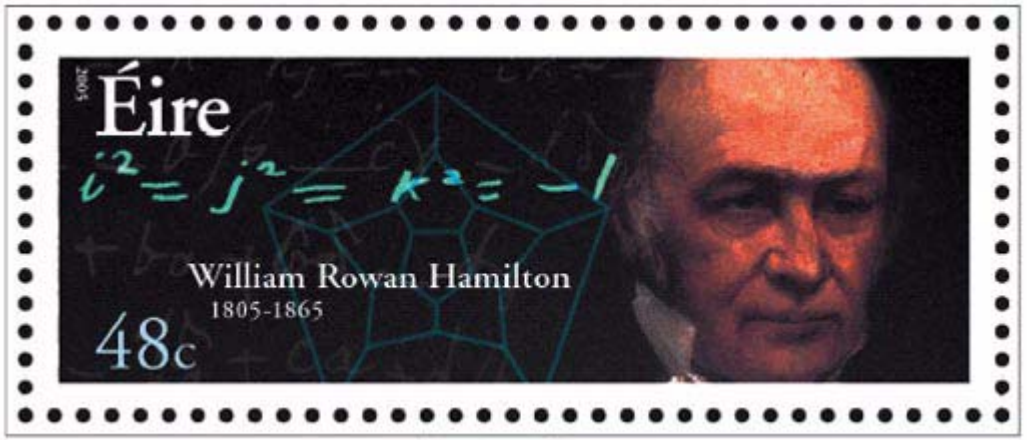


- Odd degree vertices for start and finish, otherwise not acceptable.
  - Multigraph: more than one edge between two particular vertices.
  - Degree: number of edges at a certain vertex.

Paths and Circuits

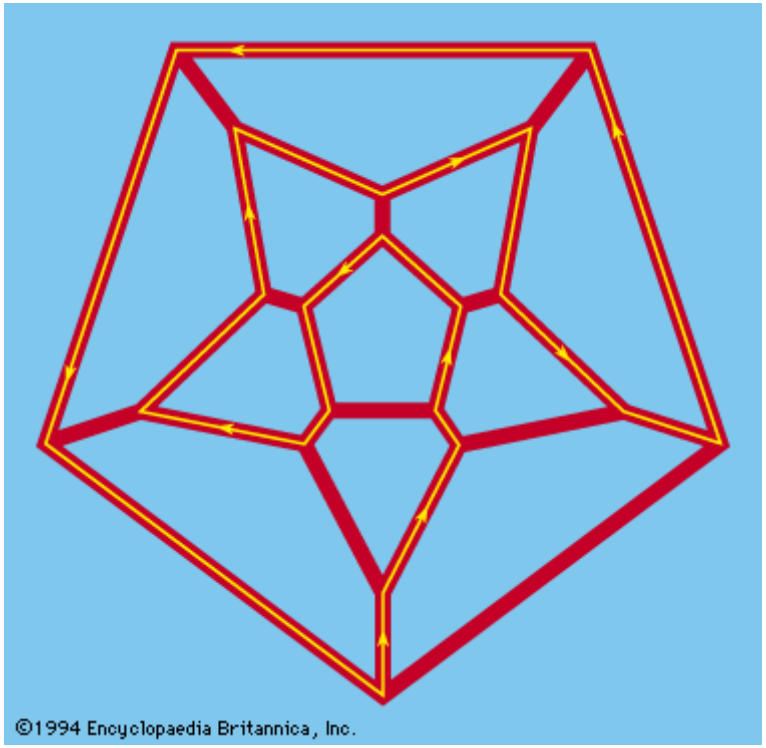
- If it is possible to start at a vertex and move along a path so as to pass along each edge exactly once, the graph has an Euler path.
- As you may recall from *Introduction to integrated circuit design*, Euler paths in a transistor representation for CMOS leads to single line of diffusion.
- If the path ends at the same vertex at which you started it is called an Euler circuit.

The Icosian Game - W R Hamilton (1857)



Hamiltonian Paths and Circuits

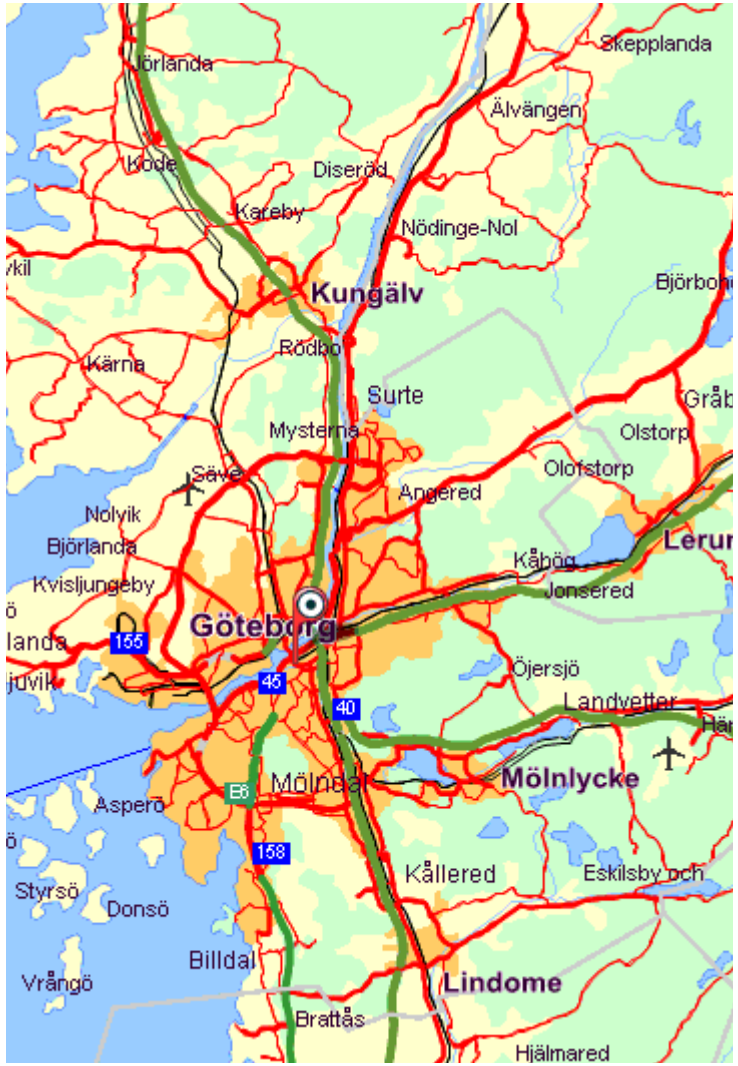
- A Hamiltonian circuit begins and ends at the same vertex, while passing through each vertex exactly once.
- For an Euler path you may visit each vertex more than once and in a Hamilton path it is not necessary to travel every edge.



- In contrast to the problem of finding out if there exists an Euler circuit in a graph, it is not practically possible to ascertain the existence of a Hamilton circuit for big problem instances.

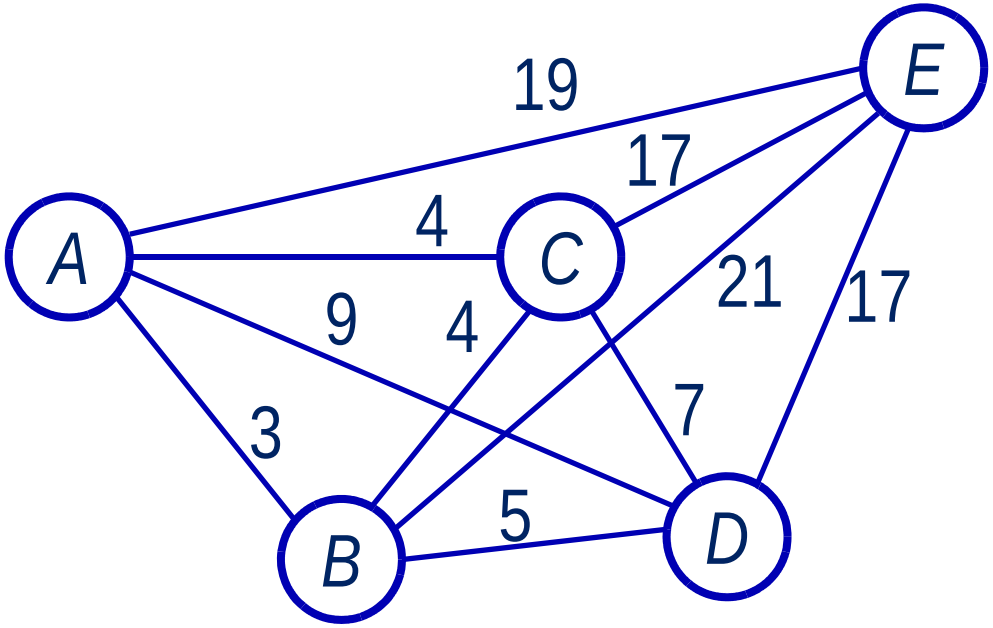
Optimization vs Decision Problems

- "What is the cheapest round-trip route that visits each city exactly once and then returns to the starting city?" This is an optimization problem.
- Compare to a decision problem where we only need to know if there exists a solution, but where we do not really need to know the solution itself: "Does a route with length under a certain distance exist?" (Yes or No.)



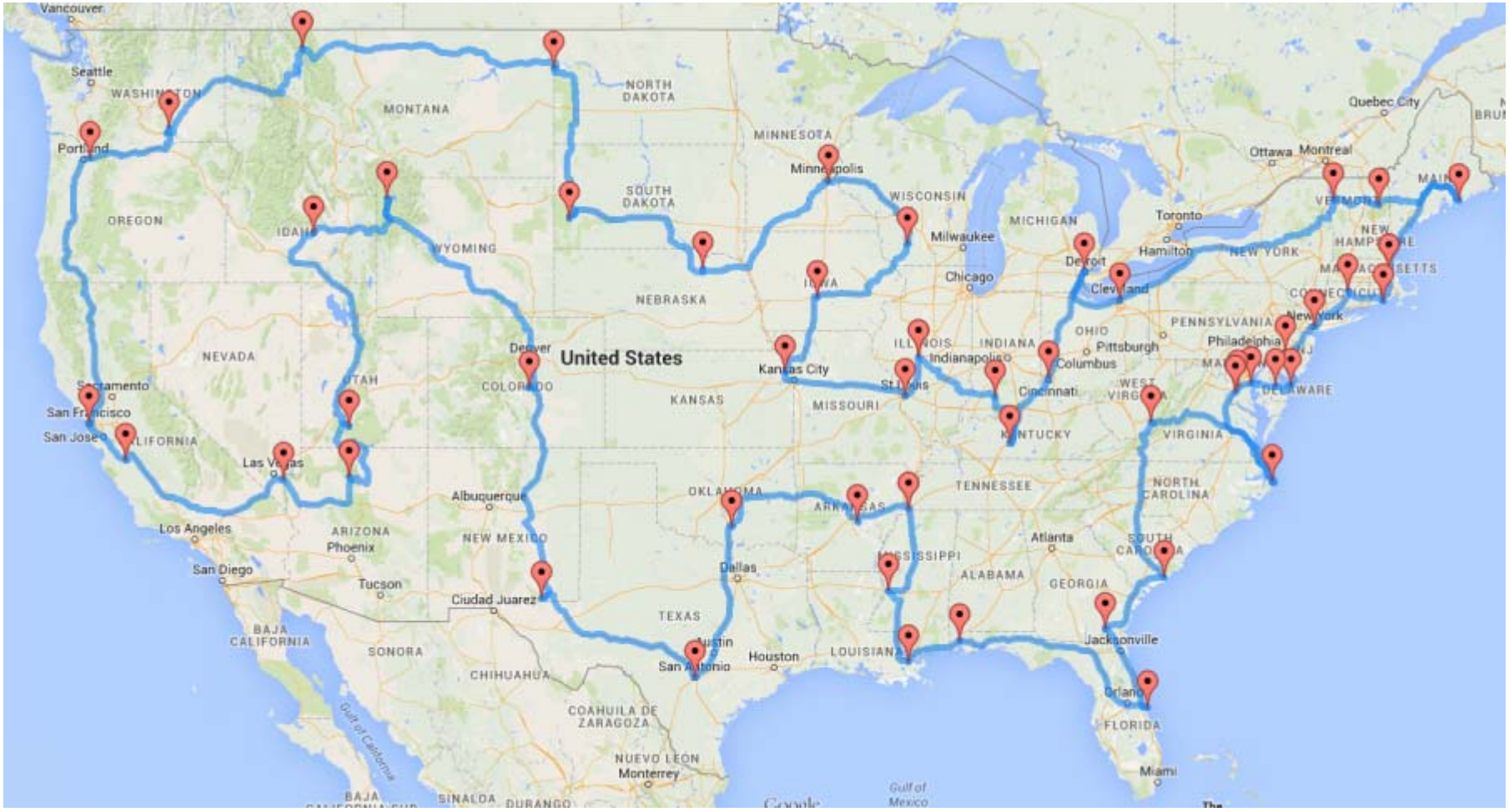
The Traveling Salesman Problem (TSP)

- In this problem we have assigned weights to the edges; a weight here means distance between two cities.
- A vertex signifies a city, so we are looking to find the Hamiltonian circuit that has the lowest cost (sum of weights).
- This is an optimization problem.



TSP Example

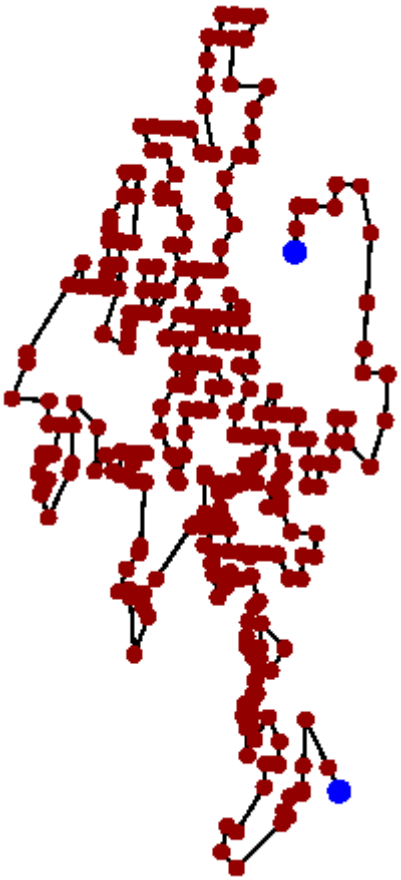
- Want to visit major landmarks in the US using the shortest route?



Source: http://www.randalolson.com/2015/03/08/computing-the-optimal-road-trip-across-the-u-s/

Another TSP Example

- Since the TSP is a so-called NP-hard problem, it is impractical to solve TSP problems that have many vertices.
- However, in 2004, the TSP of visiting all 24,978 villages and towns/cities in Sweden was solved: a tour of length (approx.) 72,500 kilometers was found and it was proven that no shorter tour exists.
  - `http://www.math.uwaterloo.ca/tsp/sweden/`
- To the right, Kungälv is the top blue vertex, while Kungsbacka is the bottom vertex of the optimal TSP solution.



Complex Problems

- Problems with polynomial dependence on input size: n^{constant}.
- Problems with exponential dependence on input size: constant^{n}.
- In general, problems with exponential dependence on input size are not feasible to compute, while those with polynomial dependence can be solved.
- NP (Non-deterministic Polynomial) problems take too long time to compute exactly, at least for large-size problems. Such problems are computationally infeasible, e.g., brute-force TSP grows with $n!$.

Complex Problems ... But Input Size Matters

- Consider an algorithm that has a worst-case complexity of $2^{n/1000}$.
- Consider another algorithm that has
  a worst-case complexity of n^{1000}.
- The first (exponentially dependent) algorithm
  is intrinsically viewed as computationally more challenging than the second (polynomially dependent) algorithm.
- However, it takes an $n > 2.4\ 10^7$ to make
  the complexity of the exponentially-dependent algorithm the larger!

NP What?

- "*The complexity class of decision problems that are intrinsically harder than those that can be solved by a nondeterministic Turing machine in polynomial time. When a decision version of a combinatorial optimization problem is proved to belong to the class of NP-complete problems, then the optimization version is NP-hard.*"

Examples:

- "Is there a Hamiltonian cycle with length less than *k*?" is NP-complete.
- However, the TSP optimization problem "What is the shortest tour?" is NP-hard.

Complexity Measures

- Computational complexity is an abstract measure to describe how difficult it is to execute an algorithm.
  - 1844 (G. Lamé): #computational steps of the Euclidean algorithm (CGD) are fewer than 5 $\log_{10} b$, where b *is* the smaller number.
  - Problem size: For the graph $G(V, E)$, the number of vertices ($|V|$) and edges ($|E|$) represent input size.
- Different complexities:
  - Worst case vs average case.
  - Time (run time) vs space (memory): Count elementary computational steps (e.g., multiplications).

Complex Design and Verification Problems

- Many optimization problems in EDA are too complex to solve.
  Here exact solutions can only be found when the problem size is small.
- One should otherwise be satisfied with suboptimal solutions found by...
  - approximation algorithms:
    they can, for example, guarantee a solution within 20% of the optimum.
  - heuristics:
    nothing can be said *a priori* about the quality of the solution.

Example: The Shortest Path Problem

- Dijkstra's algorithm finds the shortest path between v_s and v_t:

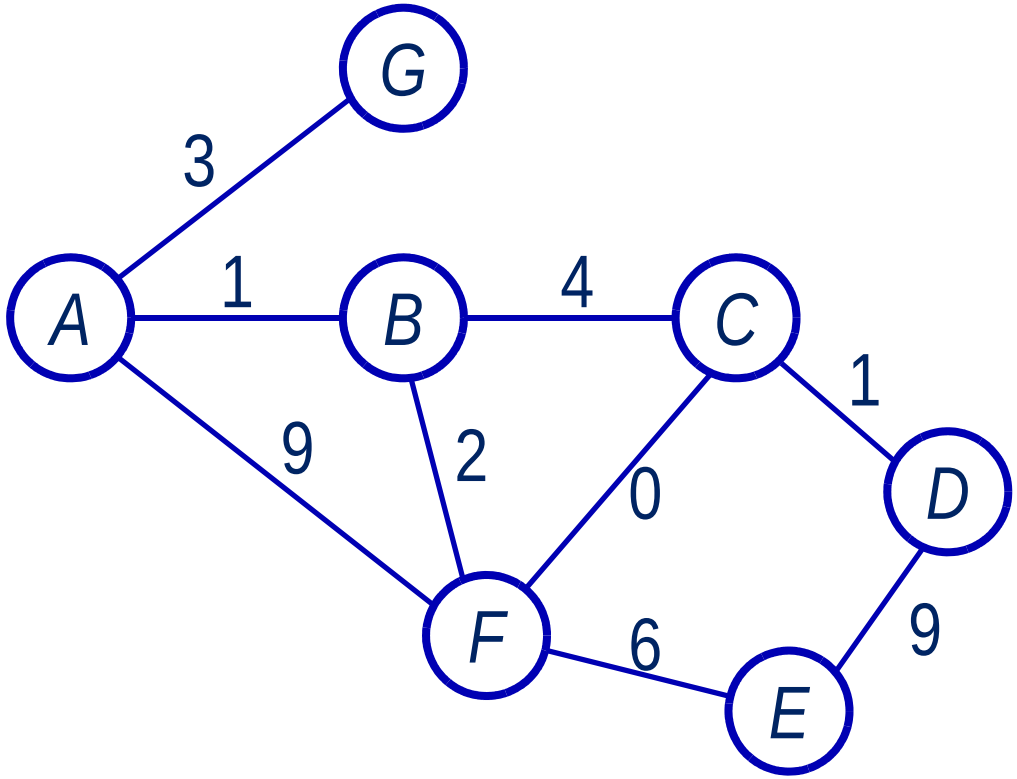
```
dijkstra(set of struct vertex V, struct vertex v_s,
struct vertex v_t)
{

set of struct vertex T;
struct vertex u, v;
V ← V \ {v_s};
T ← {v_s};
v_s.distance ← 0;
...
...
```

```
for each u ∈ V
    if ((v_s, u) ∈ E)
        u.distance ← w((v_s, u))
    else u.distance ← +∞;
    while (v_t ∉ T) {
        u ← "u ∈ V, such that ∀v ∈ V :
            u.distance ≤ v.distance";
        T ← T ∪ {u};          /* Add u to T. */
        V ← V \ {u};          /* Remove u from V. */
        for each v "such that (u, v) ∈ E"
            if (v.distance > w((u, v)) + u.distance)
            v.distance ← w((u, v)) + u.distance;
    }
}
```

Shortest Path Example 1(7)

- Find shortest path: *A* to *E*:
- We start with

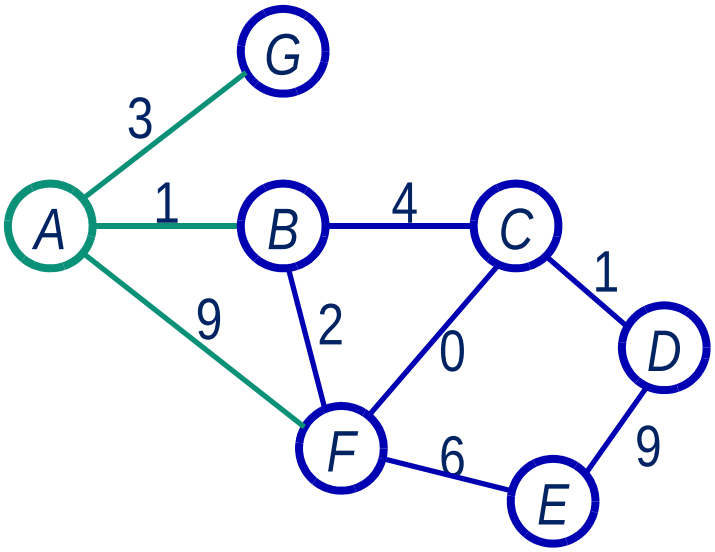
  - $T = \{\ \}$
  - $V = \{ A:(\infty, ?), B:(\infty, ?), C:(\infty, ?), D:(\infty, ?), E:(\infty, ?), F:(\infty, ?), G:(\infty, ?) \}$



Shortest Path Example 2(7)

1. We start from *A*, so we move *A* to the *T* set and update the distances to all of its (*A*'s) neighbors. *B*, *F*, and *G* all have actual paths and distances.

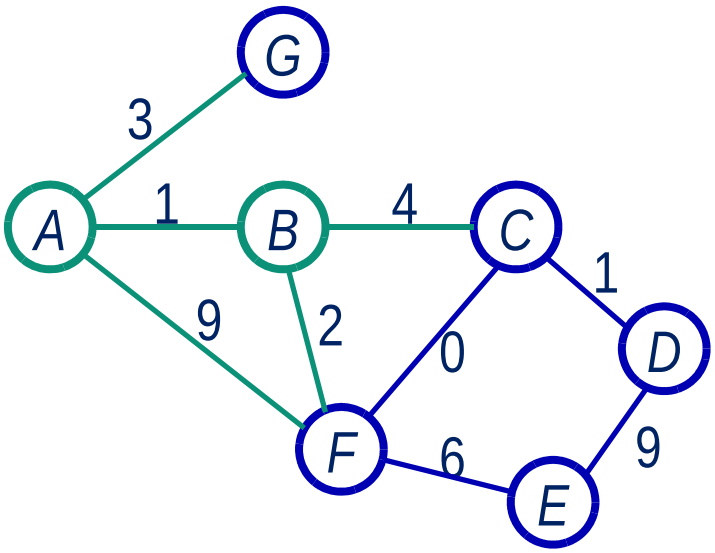
   We now have the following information :



- *T* = { *A*:(0, empty) }
- *V* = { *B*:(1, *A*), *G*:(3, *A*), *F*:(9, *A*), *C*:(∞, ?), *D*:(∞, ?), *E*:(∞, ?) }

Shortest Path Example 3(7)

2. The shortest distance is to *B*, so we move *B* to the *T* set and update its neighbors.

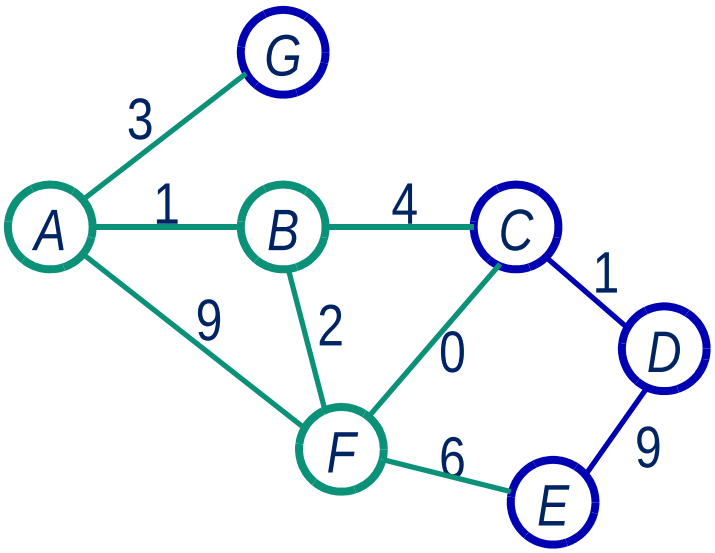


A path to *C* is known and an improved path to *F* is revealed. ($A \rightarrow B \rightarrow F$ has a cost 1 + 2 = 3).

- $T = \{ A:(0, \text{empty}), B:(1, A) \}$
- $V = \{ F:(3, B), G:(3, A), C:(5, B), D:(\infty, ?), E:(\infty, ?) \}$

Shortest Path Example 4(7)

3. *F* and *G* have equal distances:
   Still, we choose to move *F*.



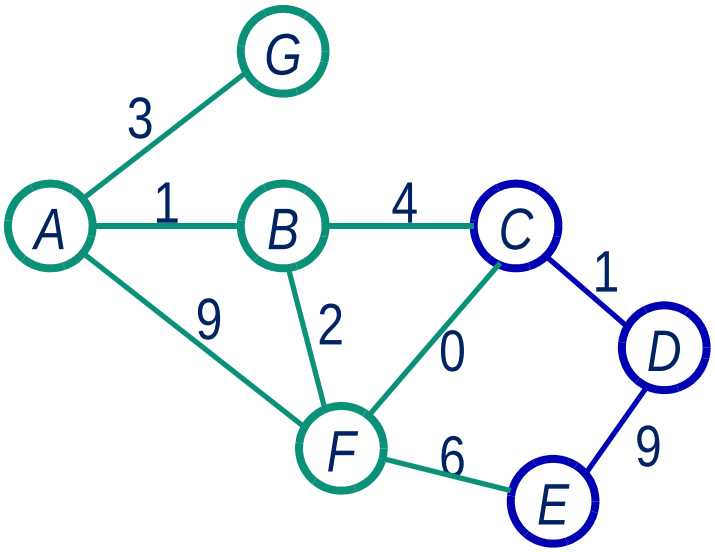
We discover a shorter path to *C*
(*A* → *B* → *F* → *C* has cost 3,
while *A* → *B* → *C* had cost 5)
and a path to *E*.

- *T* ={ *A*:(0, empty), *B*:(1, *A*), *F*:(3, *B*) }
- *V* = { *G*:(3, *A*), *C*:(3, *F*), *E*:(9, *F*), *D*:(∞, ?) }

Shortest Path Example 5(7)

4. We can then move *G* or *C* to *T*;
   they have equal costs. Now choose *G*.

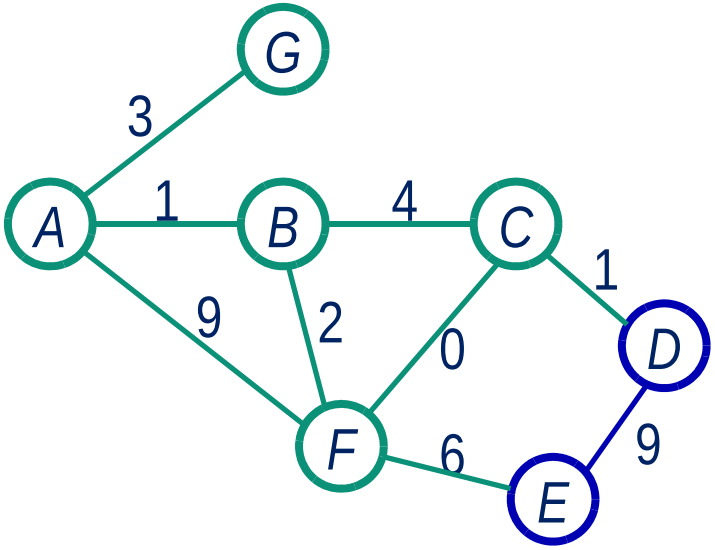
   Since *G* has no neighbors,
   there are no changes to *V*.



- *T* = { *A*:(0, empty), *B*:(1, *A*), *F*:(3, *B*), *G*:(3, *A*) }
- *V* = { *C*:(3, *F*), *E*:(9, *F*), *D*:(∞, ?) }

Shortest Path Example 6(7)

5. It is time to move *C*, which allows us to update the distance to *D*.

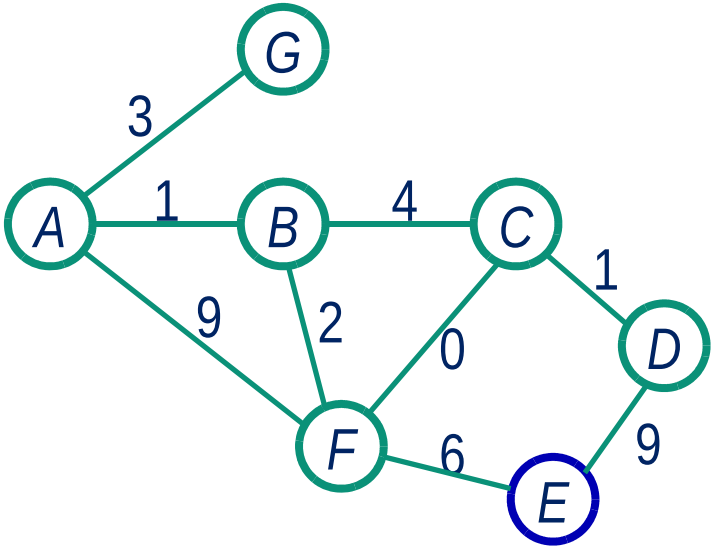


- *T* = { *A*:(0, empty), *B*:(1, *A*), *F*:(3, *B*), *G*:(3, *A*), *C*:(3, *F*) }
- *V* = { *D*:(4, *C*), *E*:(9, *F*) }

Shortest Path Example 7(7)

6. Time to move *D*.

   - *T* = { *A*:(0, empty), *B*:(1, *A*), *F*:(3, *B*), *G*:(3, *A*), *C*:(3, *F*), *D*:(4, *C*) }
   - *V* = { *E*:(9, *F*) }



7. *E* is left, move it.

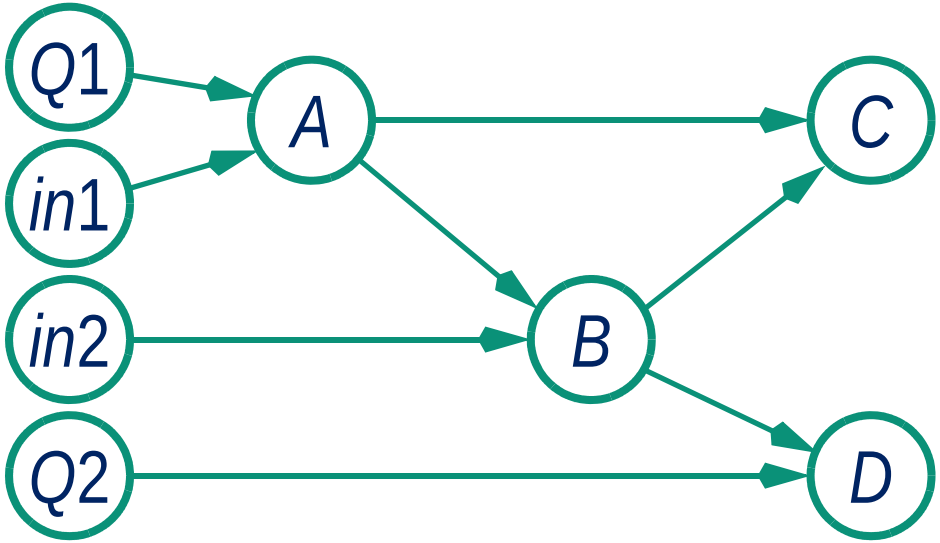
   - *T* = { *A*:(0, empty), *B*:(1, *A*), *F*:(3, *B*), *G*:(3, *A*), *C*:(3, *F*), *D*:(4, *C*), *E*:(9, *F*) }
   - *V* = { }

   Shortest path from *A* to *E* is *E*-*F*-*B*-*A* with distance cost 9.

Shortest Path Example Using Dijkstra

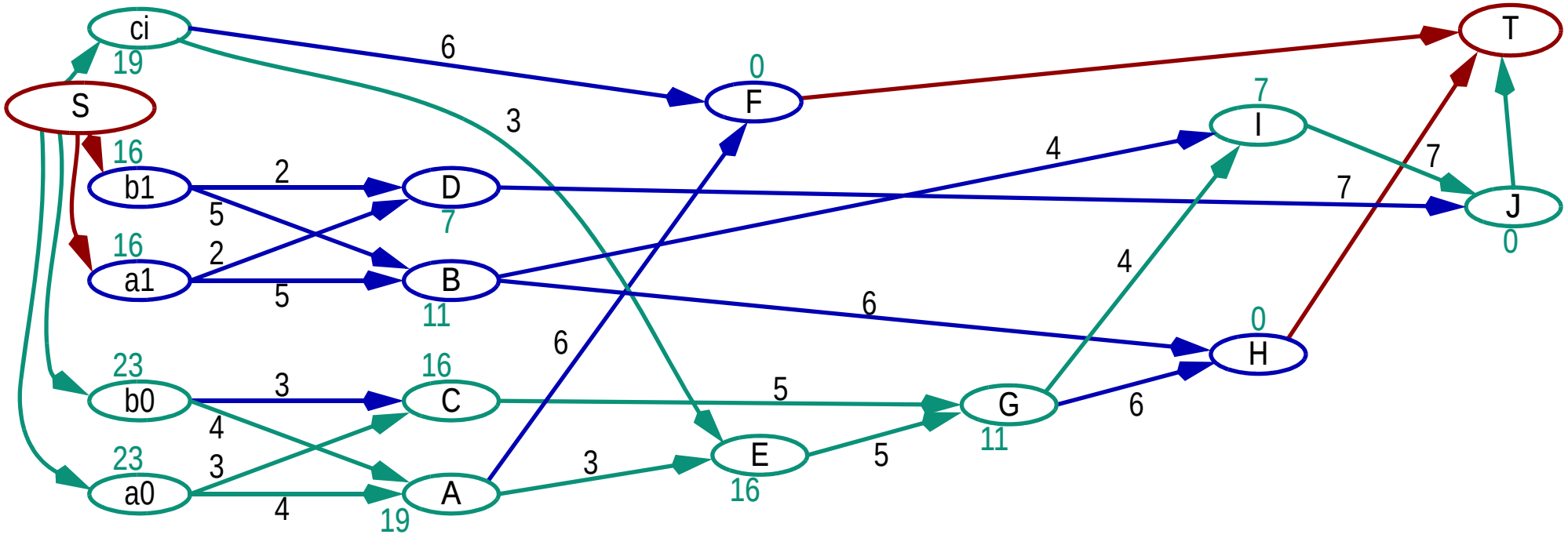
- In our example, we were unlucky to have the sought destination E as last vertex. The statement **`while`** $(v_t \notin T)$ can potentially stop the search before all vertices have been visited.
- The Dijkstra algorithm has worst-case complexity of $|V|^2 + |E|$, thanks to the skipping of paths that cannot be on the shortest path.

Algorithms in Lecture 2 ...



Depth-first search to find proper levelization.

Algorithms in Lecture 5...



Breadth-first search (from right to left),
followed by
depth-first search (from left to right)

An introduction to optimization [*flow according to Gerez*]

The Pragmatic Engineers

- EDA problems are often too complex for exact solutions; those problems are intractable.
- Tractable and intractable problems can appear very similar.
  - The shortest-path problem for undirected graphs is tractable ... solved,for example, by Dijkstra's algorithm.
  - However, the longest-path problem is intractable.

Optimization

- Optimization problem:
  finding a legal configuration
  such that its cost is minimum (or maximum).
  - "legal" implies constraints are met.
- An instance $I = (F, c)$ where ...
  - F is the set of feasible solutions, and
  - c is a cost function, assigning a cost value to each feasible solution $c : F \rightarrow R$ *(eal numbers)*
- The solution to the optimization problem is the feasible solution with optimal (minimal/maximal) cost.

Optimization Strategies

- Approximation algorithms.
  - Guaranteed to be a fixed percentage away from the optimum.
- Pseudo-polynomial time algorithms.
  - A polynomial function for the complexity, but *n* is not the problem size.
- Restriction: Work on some subset of the original problem.
- Dynamic programming (divide and conquer) [Lecture 4].
- **Exhaustive search/Branch and bound: Small problem sizes.**
- Local search.
  - **Simulated annealing** (hill climbing), genetic algorithms, etc.
- Heuristics: No guarantee of performance or quality.

Basic Optimization - LP Programming

- A production of two products P1 and P2 with ingredients I_1 and I_2.
  - P1 uses a_{11} units of I_1 and a_{21} units of I_2.
    Its unit price is c_1. Its daily production is x_1 units.
  - P2 uses a_{12} units of I_1 and a_{22} units of I_2.
    Its unit price is c_2. Its daily production is x_2 units.
  - The company cannot receive more than b_1 units of I_1
    and b_2 units of I_2 per day.
- Problem: maximize the daily revenue $c_1 x_1 + c_2 x_2$ subject to ...
  - $a_{11} x_1 + a_{12} x_2 \leq b_1, \qquad x_1 \geq 0$
  - $a_{21} x_1 + a_{22} x_2 \leq b_2, \qquad x_2 \geq 0$

LP Programming and the Integer Version

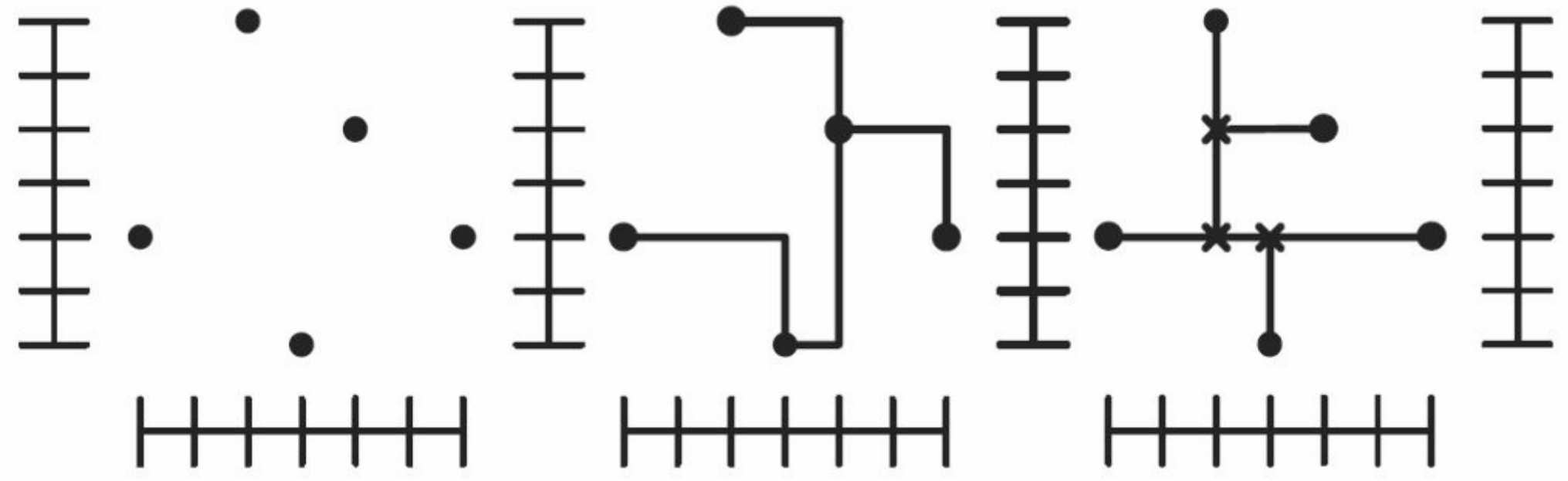
- Linear Programming - optimization in the continuous domain.
  - The LP problem can be solved using the Simplex method.
  - Simplex has exponential complexity in the worst case, but is mostly still feasible in practice.
- Integer Linear Programming - the discrete domain.
  - With the condition that variables (x_1 and x_2) are integers, the problem turns into ILP: An NP-hard optimization problem.
  - However tempting, rounding the result of LP is not possible; the solution may be infeasible or nonoptimal.

ILP FOR TSP

- The graph $G(V, E)$ with edge weights w.
- Introduce a variable x_i for each edge $e_i \in E$, $1 \leq i \leq k$.
  Here $x_i = 1$ if and only if e_i is part of the TSP solution.
- Cost function to minimize: $\sum_{i=1}^{k} w(e_i)x_i$
- Constraints to obey are ...
  - that only two edges per vertex are selected.
  - that there are no multiple disjoint tours.

Example of Approximation - Global Routing

- The minimal rectilinear spanning tree problem is tractable, while the minimal rectilinear Steiner tree is intractable.

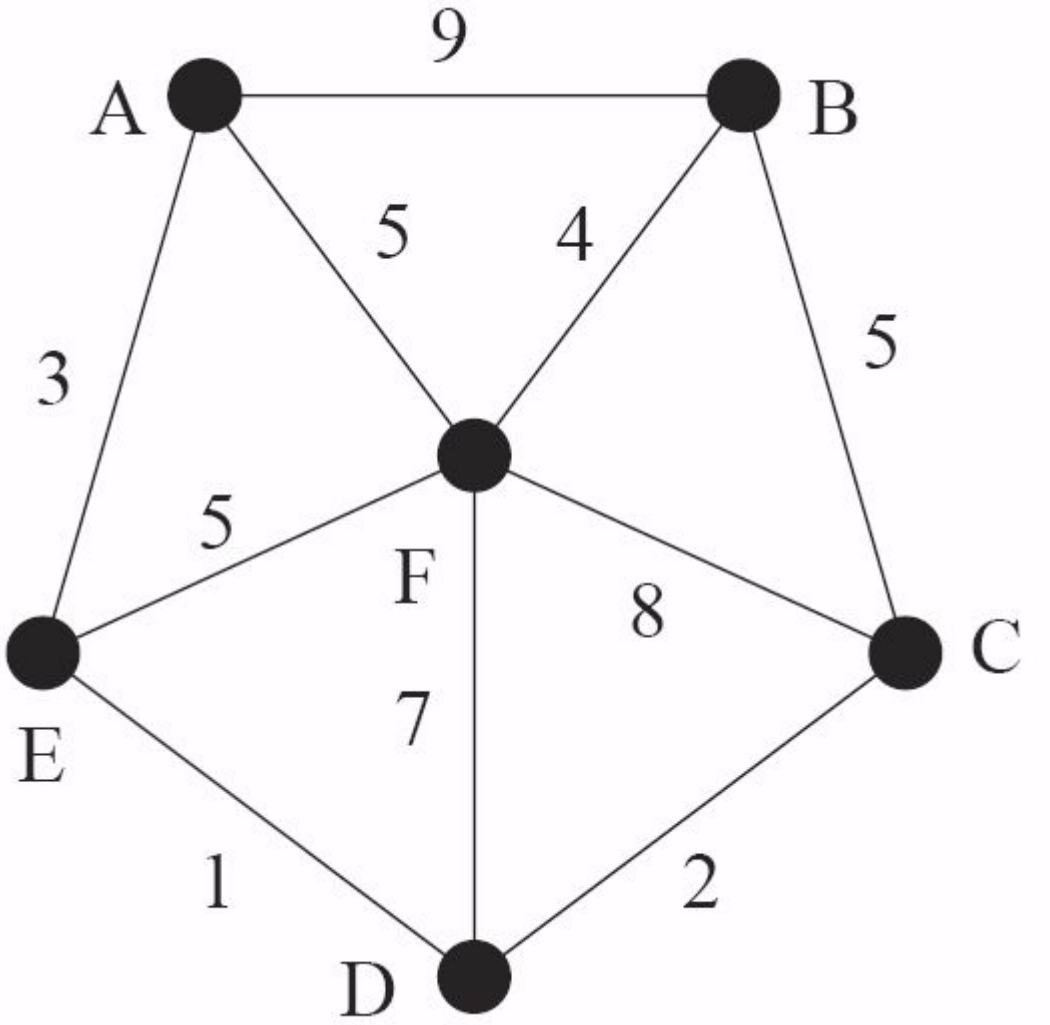


- The Minimum Spanning Tree (MST) algorithm (middle) can be an approximation for the Steiner tree problem (right).

Routing Terminology + Result

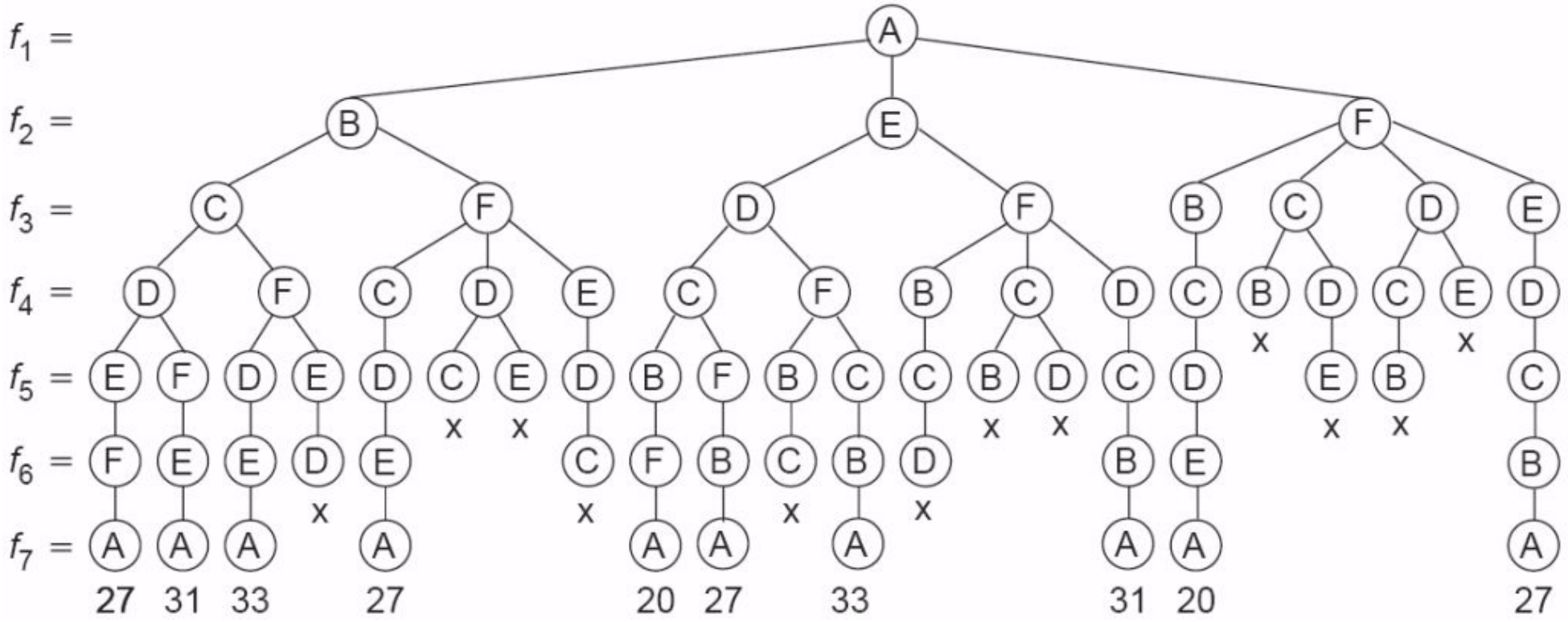
- Manhattan distance: If two points (nodes) are located at coordinates (x_1, y_1) and (x_2, y_2),
  the Manhattan distance between them is given by $d_{12} = |x_1-x_2| + |y_1-y_2|$.
- Rectilinear spanning tree: A spanning tree that connects its nodes using Manhattan paths.
- In a Steiner tree, additional points (Steiner points) are permitted to be used for the connections.
- MST approximates the Steiner tree problem solution with a quality that is less than 50% away from optimum!

Exact Solutions - The TSP Again



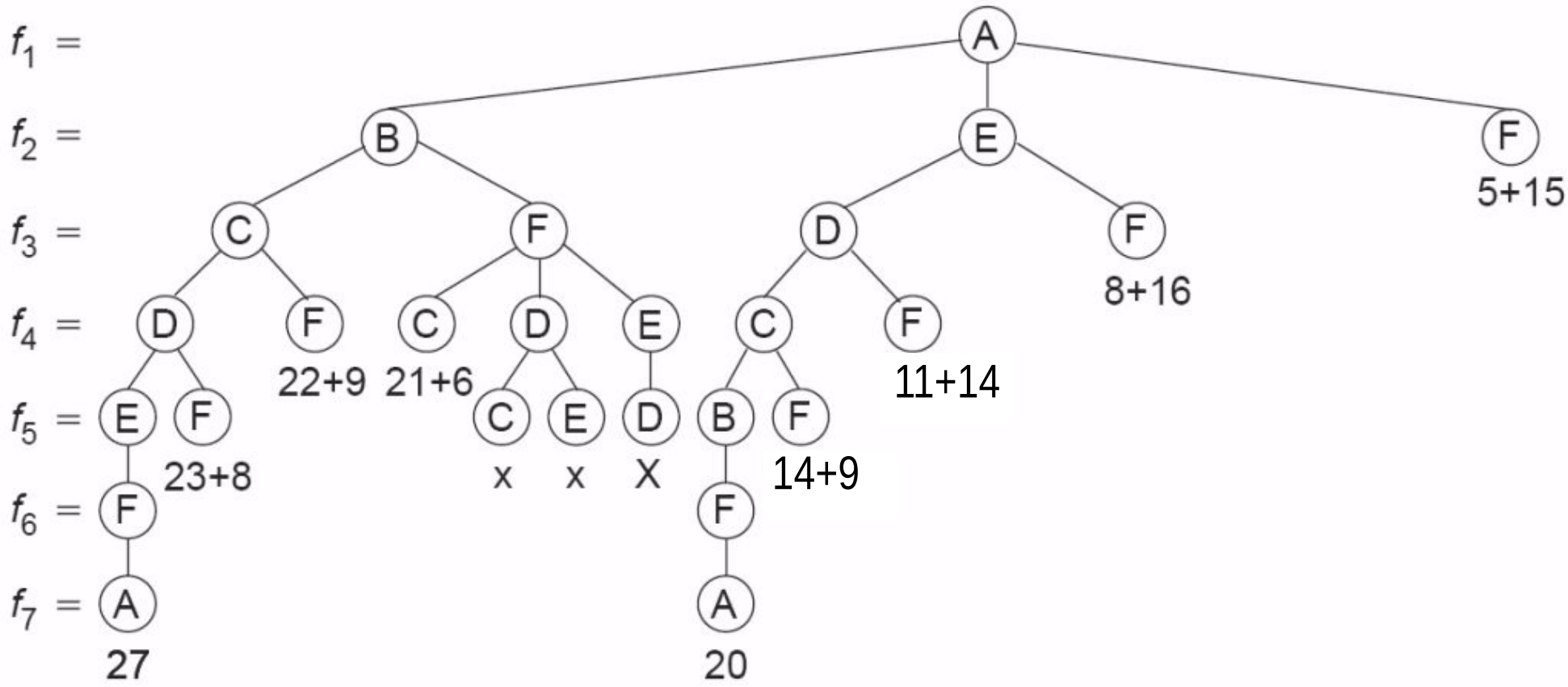
Exhaustive Search Gives Large Search Space

- The Hamiltonian circuits of the graph can be searched exhaustively.



Branch and Bound of Search Space

- After we have visited the leftmost leaf with cost 27, we compare all coming branches to this: Known path + expected path (MST).



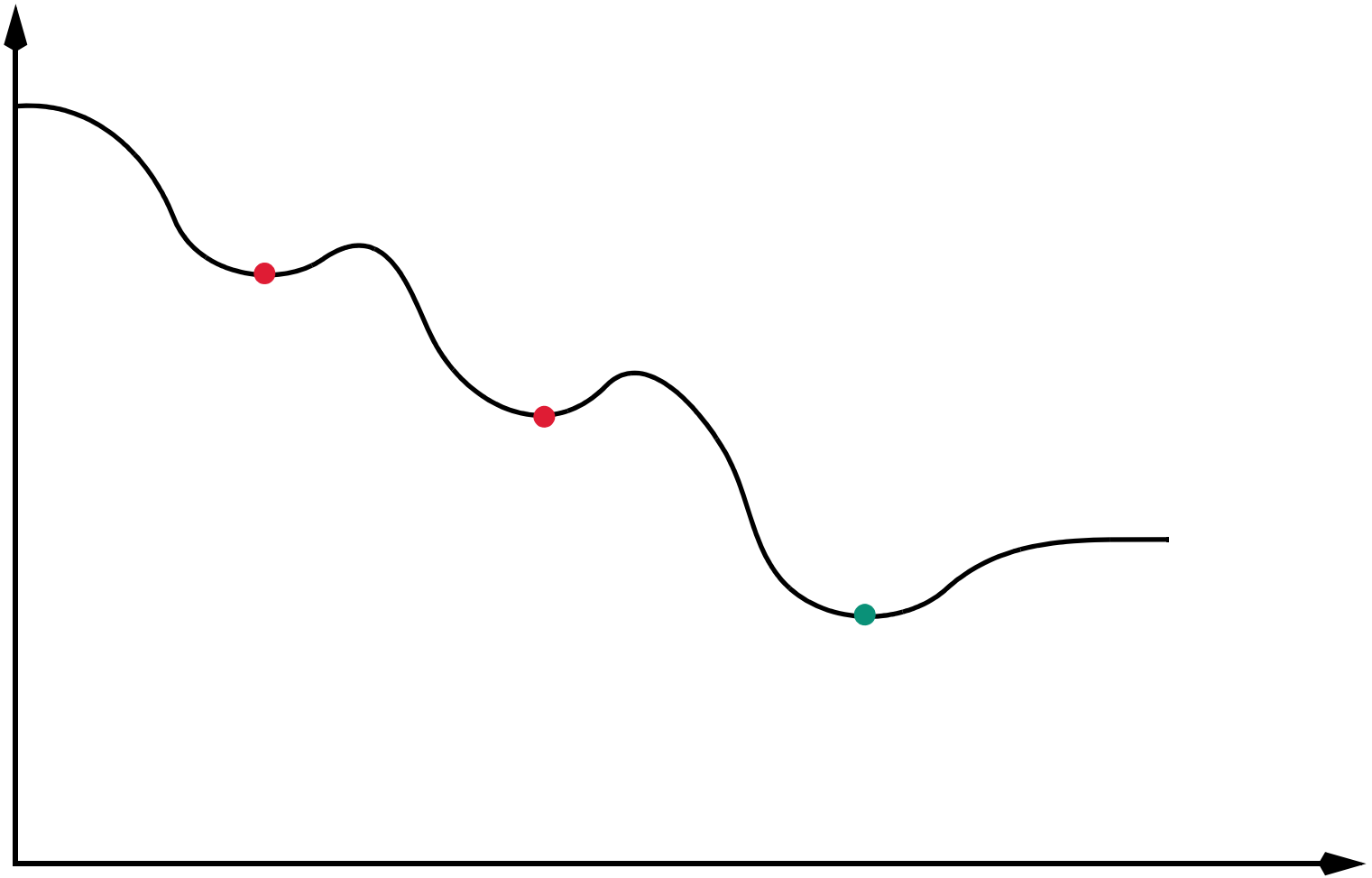
Local Search

- The simplest form of local search is gradient search.

  In a continuous representation, follow the gradient $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$;

  uphills (maximization) or downhills (minimization).
- When there exist several local optima, local search has a risk of getting stuck in a local optimum that is not the global optimum.
- Initial guess is very important.
- There are approaches that perturb the search, so there is a small probability that you make a “bad” move: Tabu search, simulated annealing and genetic algorithms.

The Weakness of Gradient Search



Idea of Simulated Annealing

- High enough temperature to ensure random state + cooling process slow enough to ensure thermal equilibrium

  $\rightarrow$

  the atoms will place themselves in a pattern that corresponds to the global energy minimum of a perfect crystal.

- Key mechanism:
  Bad moves may be accepted,
  which allows us to leave local optima!

Simulated Annealing 1(2)

- "A material cools down slowly and settles to a minimal energy state."
  - Energy $\leftrightarrow$ cost function.
  - Molecule movement $\leftrightarrow$ movement in search space.
  - Temperature $\leftrightarrow$ control parameter T.
- Move strategy for f (current position) and $g = m(f)$ (next position).
  - $\Delta c = c(g) - c(f)$.
  - If $\Delta c \leq 0$, always accept transition to g.
  - If $\Delta c > 0$, accept with a probability limit of $e^{-\frac{\Delta c}{T}}$.

Simulated Annealing 2(2)

1. Initialize: Start with a random initial condition
   and a high temperature.
2. Move: Perturb for example initial placement through a move.
3. Calculate cost: Calculate cost change due to the move made.
4. Choose: Accept or reject the move.
   Probability of acceptance depends on the current temperature.
5. Update and repeat: Update the temperature value
   (perhaps it is time to lower the temperature).
   Check if “freezing point” is reached, then quit,
   else go back to step 2.

SA Algorithm for Placement

```
begin
 temp = initial_temperature;
 place = initial_placement;
 while (temp > freezing_point) do
     while (inner_loop_criterion = FALSE) do
         new_place = PERTURB(current_place);
         ΔC = COST(new_place) - COST(current_place);
         if (ΔC < 0) then
             current_place = new_place;
         else if (RANDOM(0,1) > e^-(ΔC/T)) then
             current_place = new_place;
     temp = SCHEDULE(temp);
end;
```

SA Issues

- Cooling schedule is very important for quality of result.
  - Number of iterations in inner and outer loops.
  - Temperature update strategy.
- Optimal cooling schedule guarantees optimal solution with probability = 1, but requires infinite number of iterations.
- In practice, simulated annealing ...
  - degrades to heuristic due to nonoptimal cooling schedule.
  - is the best method known for (standard-cell) placement.
  - cannot compete with good problem-specific heuristics, if such exist.

Discrete Math and Optimization: Conclusion

- Can you solve an optimization problem exactly, do so.
- Application of approximation algorithms and heuristics usually requires understanding of context; that is, it requires experience.
- General optimization algorithms, such as simulated annealing, are generic. Do not expect too much from such algorithms; they are often used as the last resort for “inexperienced” users or “hopeless” problems.
- Read more [Ch4_Algorithms.pdf].