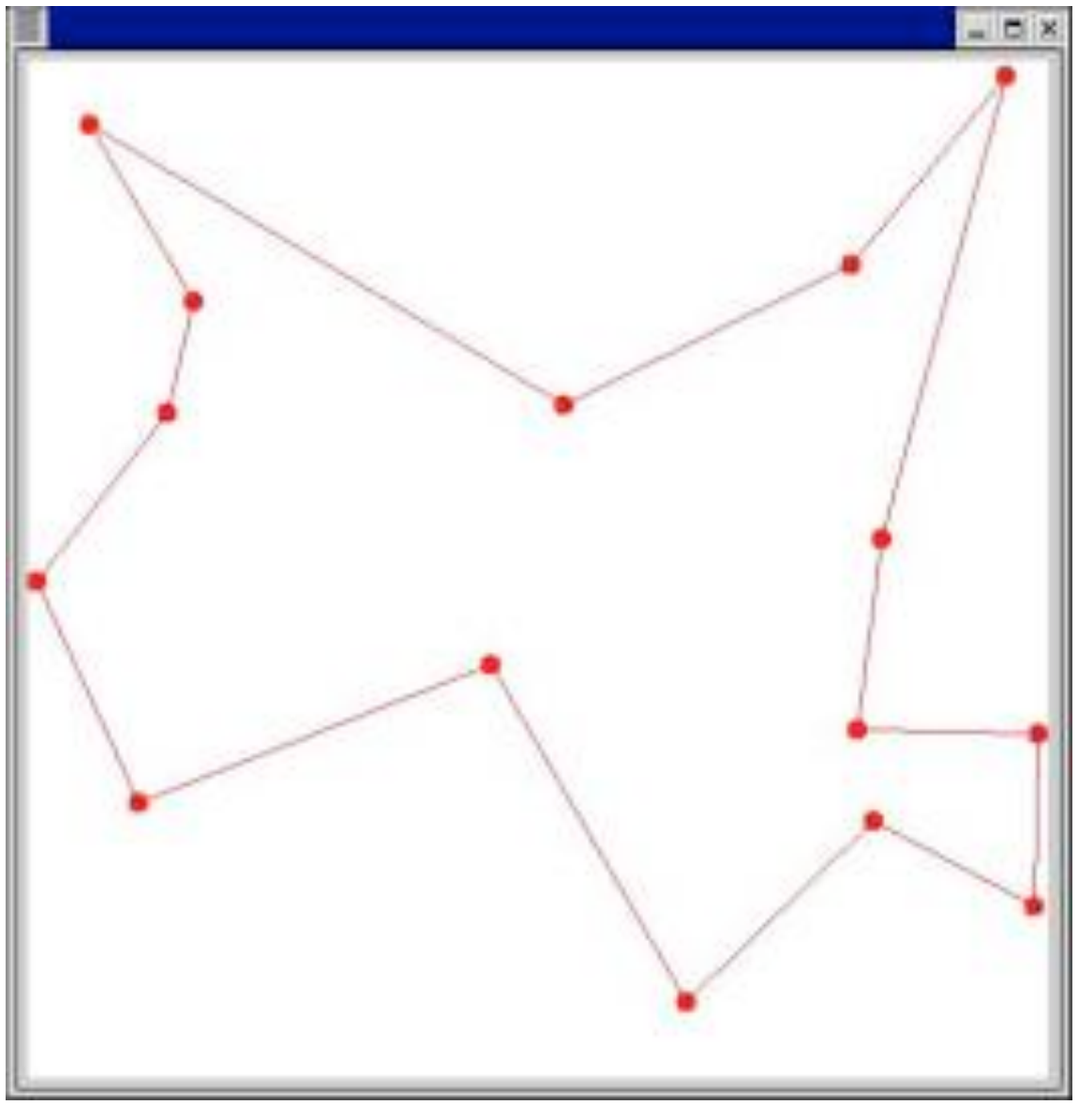
Assignment 3 (RaceTrap)

Lars Tiede (lars.tiede@uit.no), 2009-10-22

Assignment 3 - “find the shortest round-trip”



The RoadRunner survival question

- Given are locations with “food bags” for the RoadRunner in the desert
- Is the shortest possible round-trip longer than the food supply permits?
  - If so, Coyote can eat the starving RoadRunner
  - If not, the RoadRunner will survive

Summary

- Theoretical background: the *Traveling Salesman Problem*
  - quick introduction
  - conceptually easy brute force branch-and-bound algorithm
    - possible parallelization
- Assignment 3 in more detail
  - parallelize a branch-and-bound TSP solver

Traveling Salesman Problem

Formulation

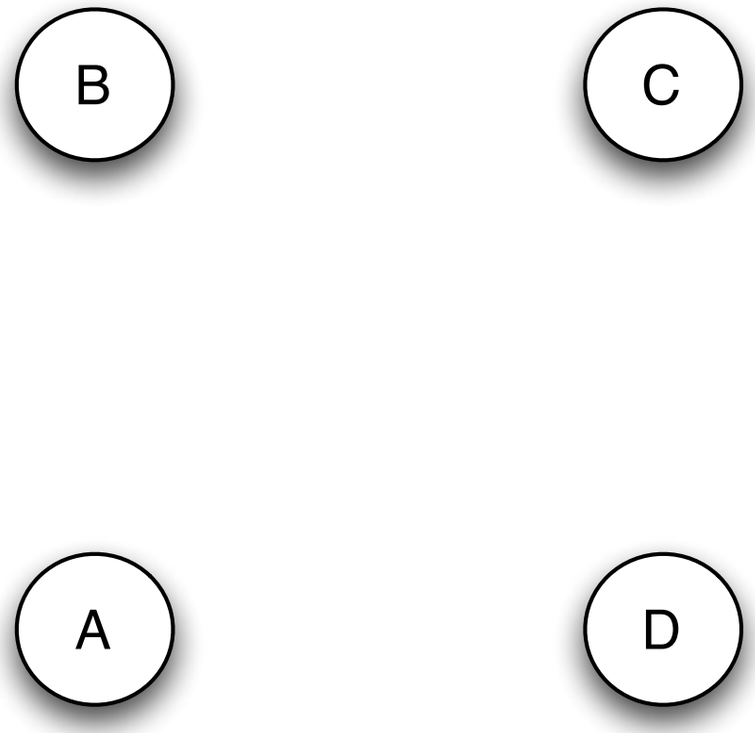
- A salesman wants to visit n cities. He wants to visit each city exactly once, and must finish where he started. In which order should he visit the cities to minimize the distance travelled?
- As a graph problem: which is the lightest hamiltonian cycle in a fully connected weighed undirected graph?
  - hamiltonian cycle: a cycle in a graph that visits each vertex exactly once. (Formal notion of a “round-trip”)

(Almost) trivial example

- Four “cities” A, B, C and D with given location coordinates

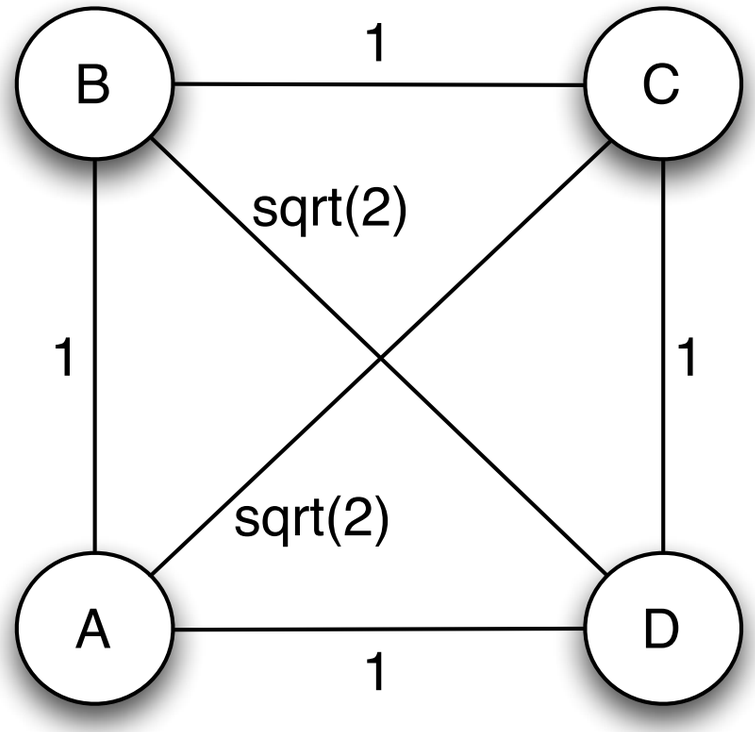
- Find shortest round-trip

- Pick any city as start / end point (we pick A)



Example: graph construction

- Make a graph from the "geometrical" information
  - simple euclidean distances
    - euclidean distance is symmetric -> undirected graph
  - fully connected
- Now find the shortest hamiltonian cycle



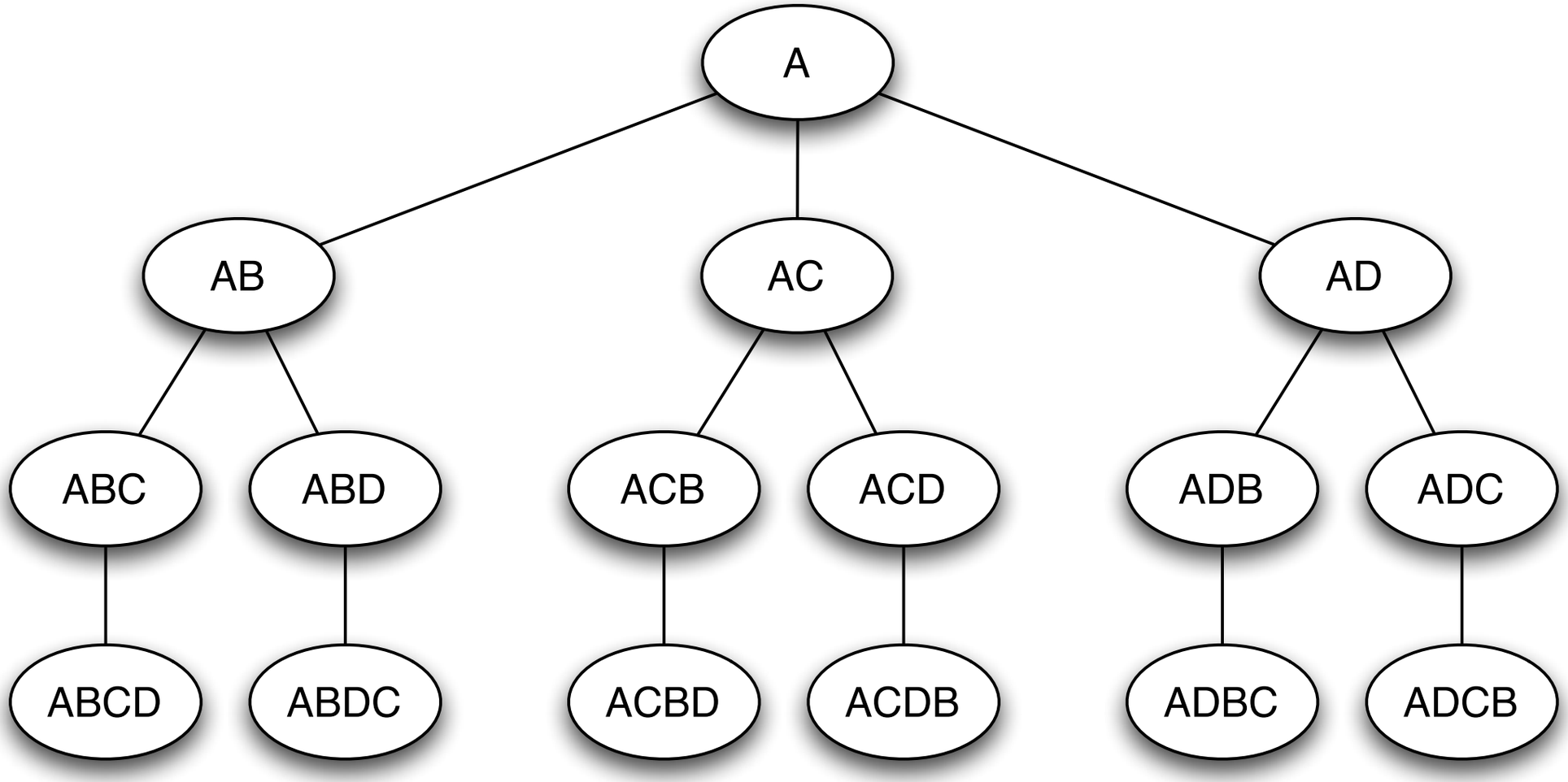
Simple brute force algorithm idea

- Assumption: to determine which route is the shortest, one must *explore them all*

- To do that, find a way to enumerate (and compute) all hamiltonian cycles ("round-trip routes")
  - all hamiltonian cycles in our case: all permutations of sequences containing all cities

- Among them, pick the shortest one

Example: search space



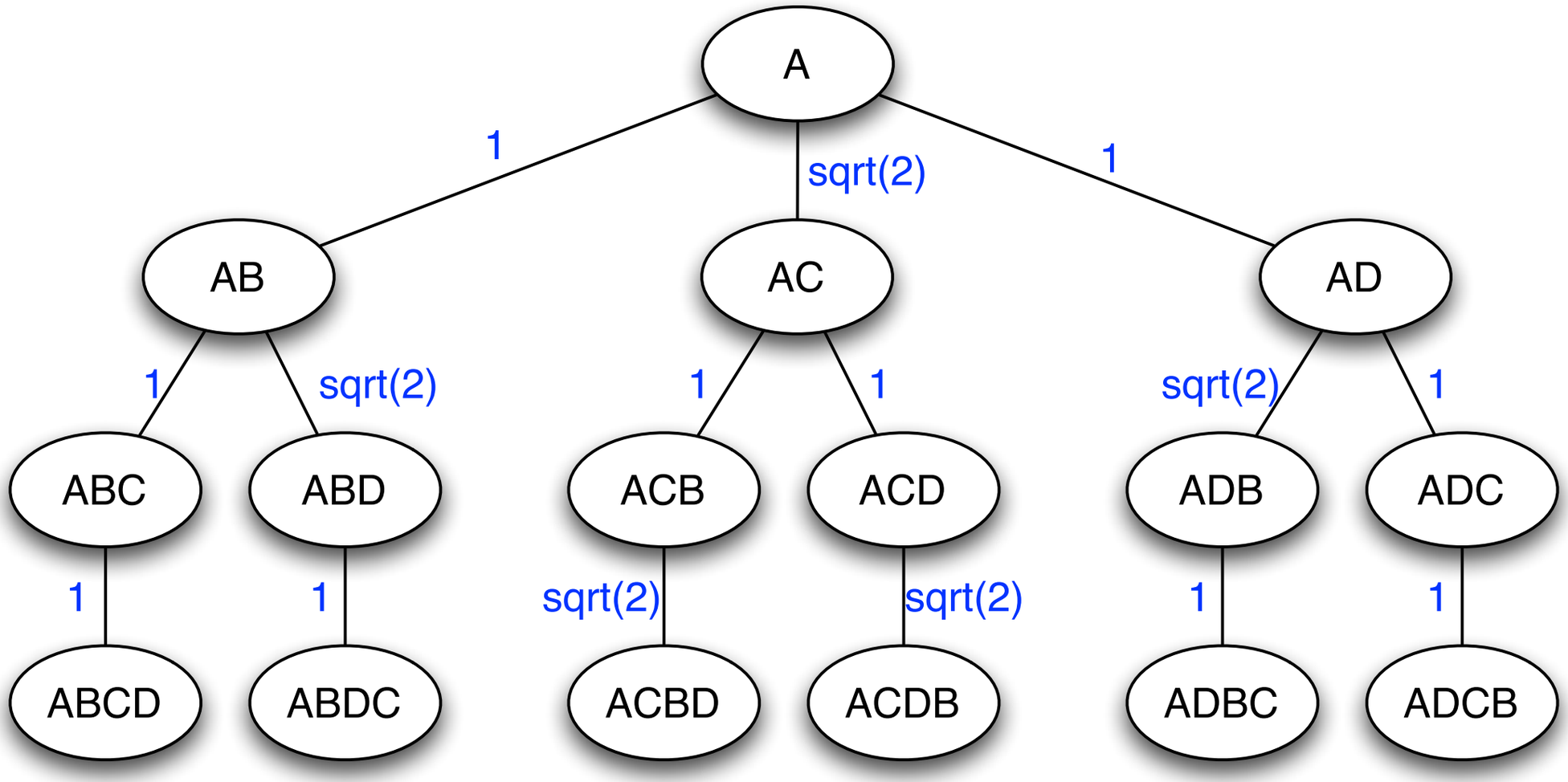
Search space construction

- *Search space* can be formulated as a tree (last slide)
  - a node contains the information on which route has been taken so far in that search branch
    - root of the tree is the start city (choose any)
- children are defined by all possible continuations of a route ("which city could be next")
- leafs contain sequences of all cities, thereby complete routes
  - the set of leafs is the set of all possible complete routes

Simple brute force algorithm

- Traverse through the search space tree (depth first) and keep track of the best found complete route
- Once the tree has been fully traversed, the best route is known

Example: route lengths



Simple brute force algorithm (sequential)

```
best_route = (empty list);
best_route_length = INFINITY;

def recursive_search(route_so_far:list, remaining_cities:list) :
    if remaining_cities is not empty :                  # recurse!
        for city in remaining_cities :
            extended_route = route_so_far + city
            then_remaining = remaining_cities - city
            recursive_search(extended_route, then_remaining)
    else :                                              # got complete route!
        length = calculate_total_route_length(route_so_far)
        if length < best_route_length :
            best_route = route_so_far
            best_route_length = length

# for the example, run: recursive_search([A], [B,C,D])
```

Optimization

- Branch-and-*bound*: do not research further into routes whose completions can not yield a better solution than the best already known solution
  - Given an incomplete route, compute a lower bound for any completion of that route. If (lower bound for route length >= best known route length), then disregard all such completions.
    - Trivial lower bound: completions have at least length 0.
    - Easy lower bound: completions have at least the length of the shortest distance between the start / end city to any other city

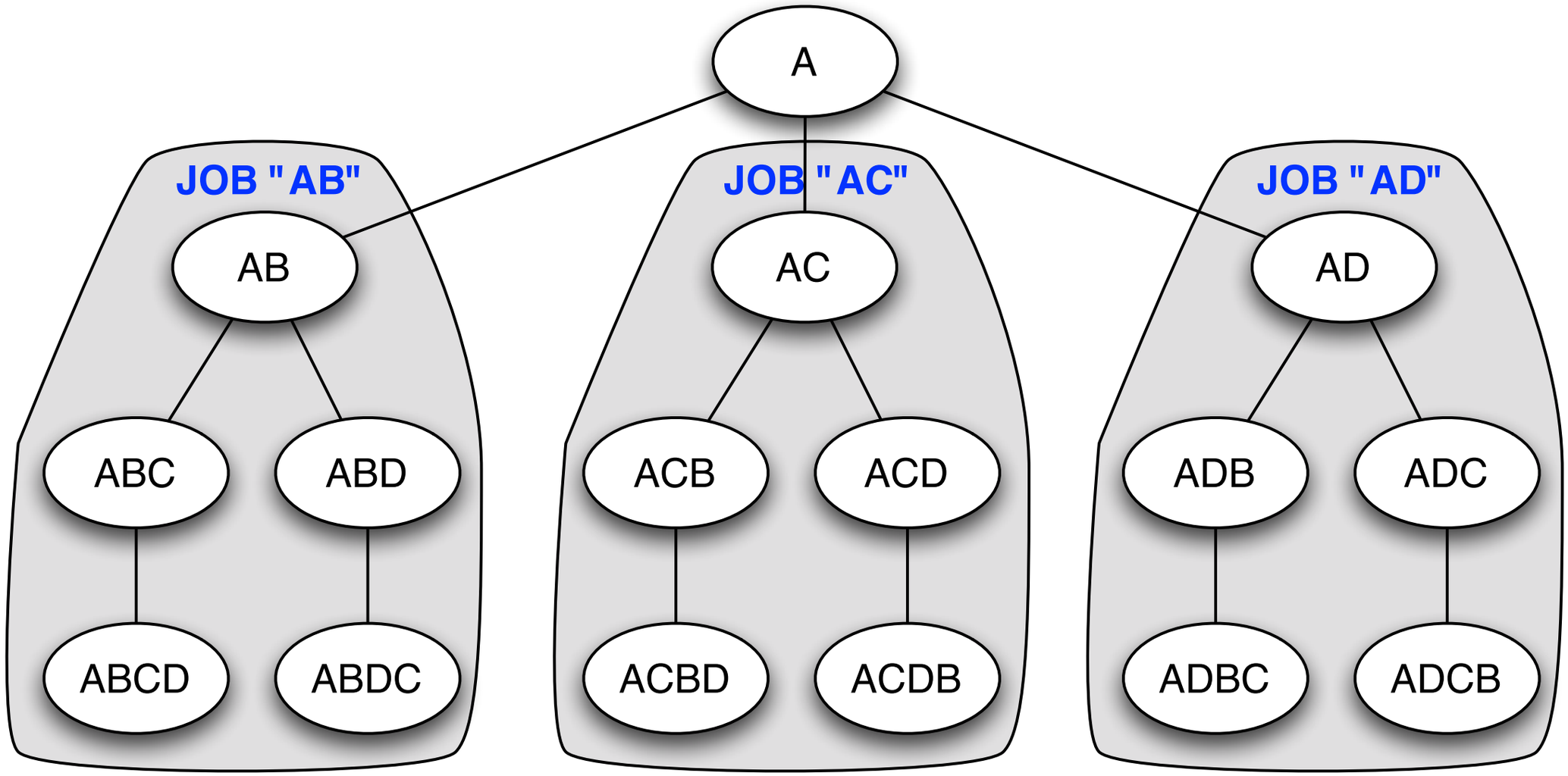
More possible optimization "hooks"

- Exploit properties of our TSP graph
  - graph is undirected (routes ABCDA and ADCBA have same length): "symmetric" TSP
  - graph weights are metric distances -> triangle inequality holds: "euclidean" or "planar" TSP

Parallelizing the simple brute force algorithm

- General idea: partition the search space and let each process search for the best solution on its own partition
- Here: partition the search space tree into subtrees and let parallel processes work on those subtrees
- Thus, we can understand any particular node in the search space tree as a *job* description
  - the job is to explore the tree rooted by that particular node

Search space partitioning



Simple brute force algorithm (parallel)

- One of several possibilities: simple master-worker scheme with central job pool
- Master generates all possible route prefixes with length k as jobs
  - determining how k should be set is an interesting issue!
- Workers get jobs from the pool, search for the shortest route completion, and send back their best search result

Optimizations for the parallel algorithm

- Like for the sequential case, but interesting issues here:
  - when and how should workers send new best routes to the master? Immediately? Or only at job completion?
  - when and how should the master send new best routes to (busy?) workers? Immediately? Or only at job dispatch?
  - interesting tradeoffs: communication cost vs. computation speedup, implementation complexity vs. performance gain
    - I'm looking forward to read about that in your reports :)

Complexity of TSP

- computationally: NP-hard problem
  - (n-1)! possible routes, and no analytical solution
    - 11 cities: 3,628,800 routes.
    - 15 cities: 87,178,291,200 routes.
- exact algorithms (guaranteeing an optimal solution):
  - brute force branch-and-bound: O(n!)
  - dynamic programming: O(2^n)
- Due to complexity, exact algorithms impractical for considerable problem sizes. Heuristics often used.

Assignment 3 technicalities

Food bag setup

- The text file "route.dat" contains 31 food bag coordinates
- First line in the file contains only one number: the number of food bags to use
  - Set to 13 now
  - The RoadRunner survival question has to be answered with 15 food bags
  - Play with it to see how the algorithms scale
- Function ReadRoute() reads route from that file

Algorithm

- Sequential recursive branch-and-bound algorithm given
  - uses the trivial lower bound estimate
  - uses a more efficient way to represent route prefix and remaining food bags
    - one array containing both. For n cities and a route prefix of length k, all (n-k) children are formed by swapping the (k+1)st array member with all remaining food bags
    - Example: [ABC | DEF]'s children are: [ABCD | EF], [ABCE | DF], and [ABCF | ED]

Requirements

- Make it run on any number of CPUs
- Retain the branch-and-bound nature of the algorithm
  - If you want to implement another algorithm, feel free to contact us!
- Analyze speed-up and scaling

Optionals

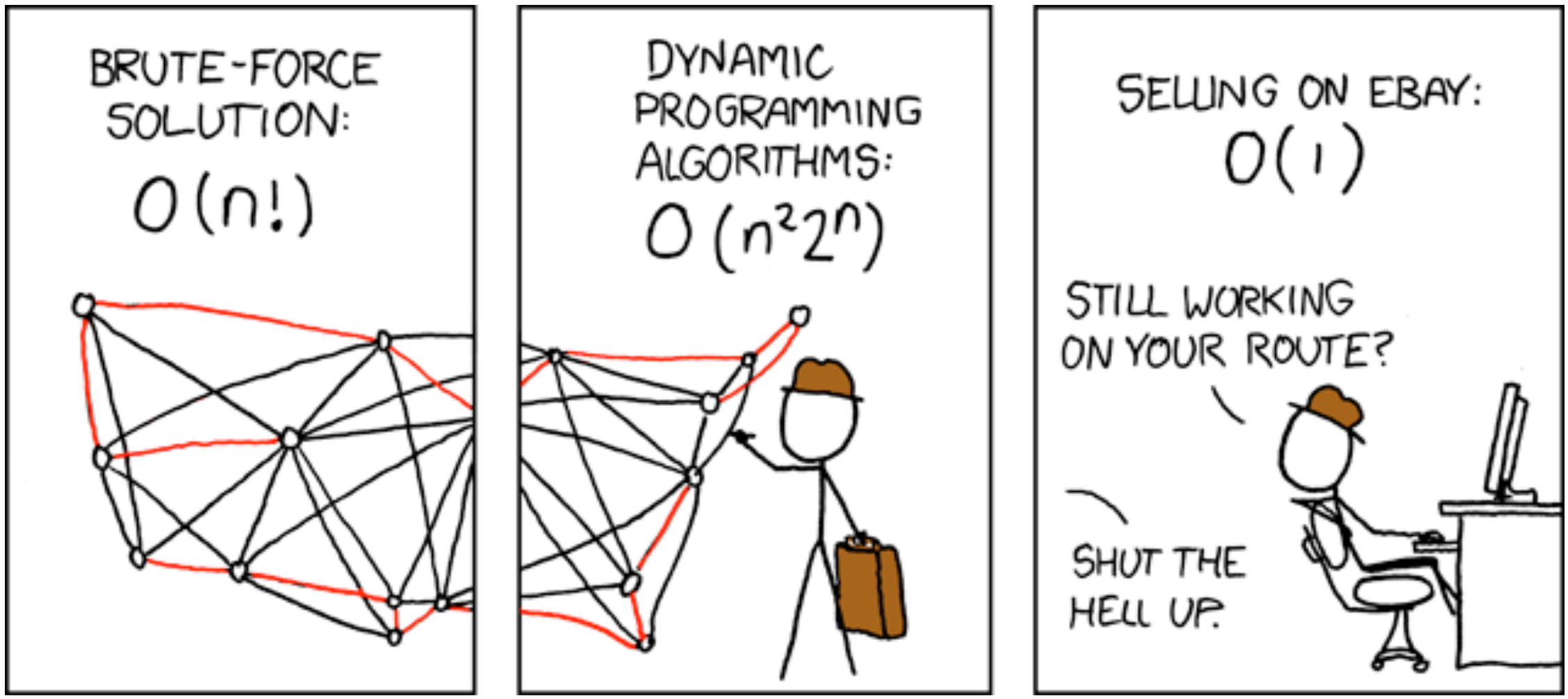
- Optimize more!
  - for example better bound estimations
- Different job organization
  - Centralized job pool might not be the optimal solution
  - How long should those route prefixes for job descriptions be?
    - Is it possible / desirable to do those distributed as well?

Hand-In

- Deadline: November, 5th, noon
- Start in time!
- Write decent reports (takes time, yes)

Further reading on TSP

- http://en.wikipedia.org/wiki/Travelling_salesman_problem
  - Decent starting point for extensive reading on TSP and related problems
- http://members.pcug.org.au/~dakin/tsp.htm
  - Easier to read than the wikipedia article
- http://www.tsp.gatech.edu/index.html
  - *Much* information!
  - The "applications" section is a short nice read



http://xkcd.com/399/