CSCU9YE - Artificial Intelligence

Lecture 5: Local Search and Metaheuristics

Prof. Gabriela Ochoa, University of Stirling

Search in Artificial Intelligence

Two types of search algorithm

Search for paths	Search for solutions
• Finding a set of actions that will bring us from an initial stat to a goal stat • Relevant to AI • **Algorithms**: depth first search, breadth first search, branch and bound, A*, Monte Carlo tree search.	• Find a solution in a large space of candidate solutions • Relevant to AI, Optimisation, Operational Research • **Algorithms**: metaheuristics, evolutionary algorithms

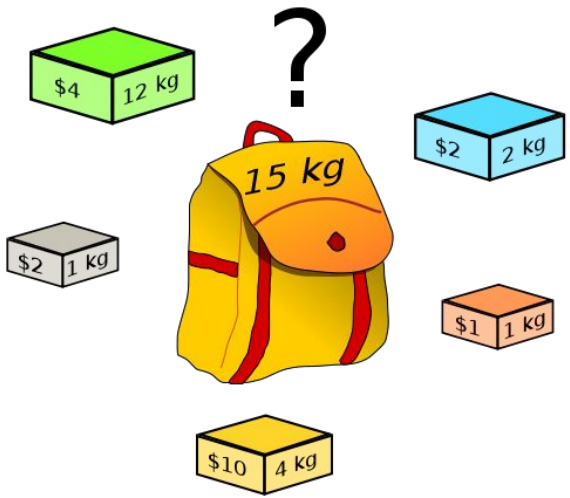
Content

1. **Optimisation problems**
   - Optimisation & search
   - Example problems (Knapsack, Travelling Salesperson, others)
2. **Optimisation methods**
   - Constructive Heuristics
   - Metaheuristics
     - Single-point algorithms
     - Population-based algorithms

Optimisation problems

- Definition (minimisation problem): c *(S,f)*
  - Given
    - a search space *S*, feasible solutions
    - an objective function *f: S -> R*
  - Find s^* in *S* such that
    - $f(s^*) \leq f(s)$ for all *s* in *S*
    - s^* is the global optimum
- Large scale/complex optimisation problems in many areas of science and industry: Transportation, Logistics, Finance, Design, Biology, Environment

The knapsack problem



Input

Capacity K

n items with weights w_i and values v_i

Goal

Output a set of items s such that

the sum of weights of items in s is at most K

and the sum of values of items in s is maximised

Knapsack, full enumeration

(only possible for very small instances)

Items	Value	Weight	Feasible?
000	0	0	Yes
001	8	5	Yes
010	12	10	Yes
011	20	15	No
100	5	4	Yes
101	13	9	Yes
110	17	14	No
111	25	19	No

```
3
1  5  4
2 12 10
3  8  5
11
```

What do we mean by random search?

Attempting several solutions at random, and keeping the best found.

```
procedure random-search
begin
  s = random initial solution solution
  repeat
     evaluate solution (s)
     s' = random solution
     if evaluation(s') is better than evaluation(s)
     s = s'
  until stopping-criterion satisfied
  return s
end
```

The stopping criterion can be a fixed number of iterations.

What is a heuristic?

An optimisation method that tries to exploit problem-specific knowledge, for which we have no guarantee to find the optimal solution

Construction

- Search space: partial candidate solutions
- Search step: extension with one or more solution components
- Example in Knapsack: Greedy construction

Improvement

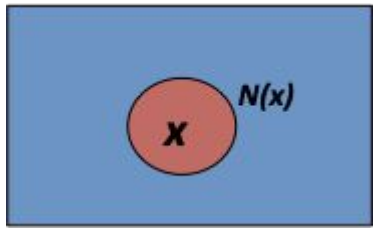
- Search space: complete candidate solutions
- Search step: modification of one or more solution components
- Example in Knapsack: hill-climbing with bit-flip

The notion of neighbourhood

- Region of the search space that is “near” to some particular point in that space
- Defined in terms of a move operator
- Binary representation
  - 1-bitflip mutation: flipping a single bit in the given bit string
  - For strings of length *n*, every solution has *n* neighbours
    - Solution: 1 1 0 0 1
    - Neighbours:

0 1 0 0 1
1 **0** 0 0 1
1 1 **1** 0 1
1 1 0 **1** 1
1 1 0 0 **0**

A search space *S*, a potential solution *x*, and its neighbourhood *N(x)*



Defining neighbourhoods

Binary representation

1-flip: Solutions generated by flipping a single bit in the given bit string

- If the string length is *n,* how many neighbours each solution has?
- Example: 1 1 0 0 1 ➔ 0 1 0 0 1

2-flip: Solutions generated by flipping two bits in the given bit string

- If the string length is *n,* how many neighbours each solution has?
- Example: 1 1 0 0 1 ➔ 0 1 0 1 1

k-flip: can be generalised for larger k, $k < n$

Hill climbing (first Improvement, random mutation)

```
procedure first-hill-climbing
begin
  s = random initial solution solution
  repeat
     evaluate solution (s)
     s' = random neighbour of s
     if evaluation(s') is better than evaluation(s)
     s = s'
  until stopping-criterion satisfied
  return s
end
```

The stopping criterion can be a fixed number of iterations

Implementation: binary representation

- 1-flip mutation: flipping a single bit in the given bit string
- For strings of length *n*, every solution has *n* neighbours
- Example: 1 1 0 0 1 → **0** 1 0 0 1,
- Python implementation

```
def mutation(sstr):
        pos = randint(0, SOLUTION_LEN-1)
        sstr[pos] = 0 if sstr[pos] else 1
        return sstr
```

```
# Initialise a random solution
def initialise():
        return [randint(0, 1) for i in range(SOLUTION_LEN)]
```

Hill climbing algorithm (Best Improvement)

```
procedure best-hill-climbing
begin
   s = random initial solution solution
   repeat
      evaluate solution (s)
      s' = best solution in the Neighbourhood
      if evaluation(s') is better than evaluation(s)
      s = s'
   until s is a local optimum
   return s
end
```

Hill-climbing methods

Weaknesses

- Usually terminate at solutions that are local optima
- No information as to how much the discovered local optimum deviates from the global (or even other local optima)
- Obtained optimum depends on starting point

Advantages

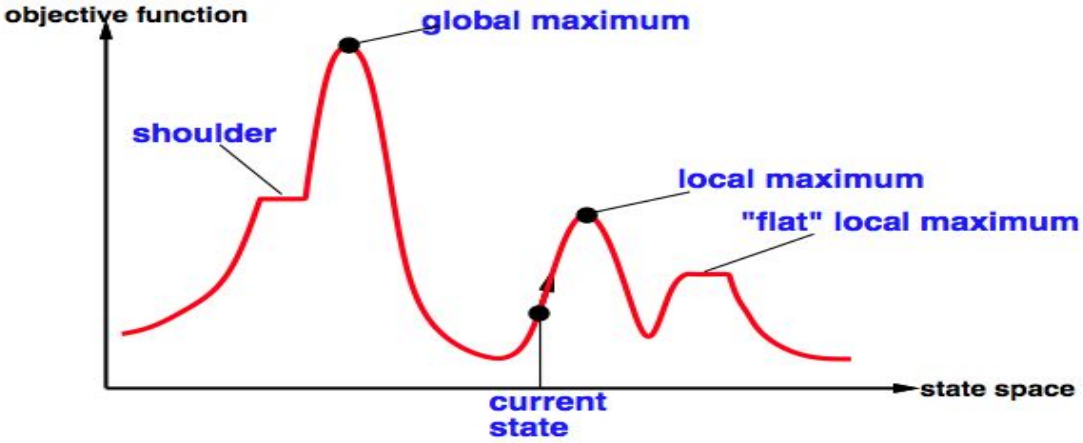
Very easy to apply. Only needs

- A representation
- The evaluation function
- A neighbourhood

Hill-climbing search

Problem: depending on initial state, can get stuck in local maxima

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves ☺ escape from shoulders ☹ loop on flat maxima

Constraint handling

Not trivial to deal with constraints. Alternatives:

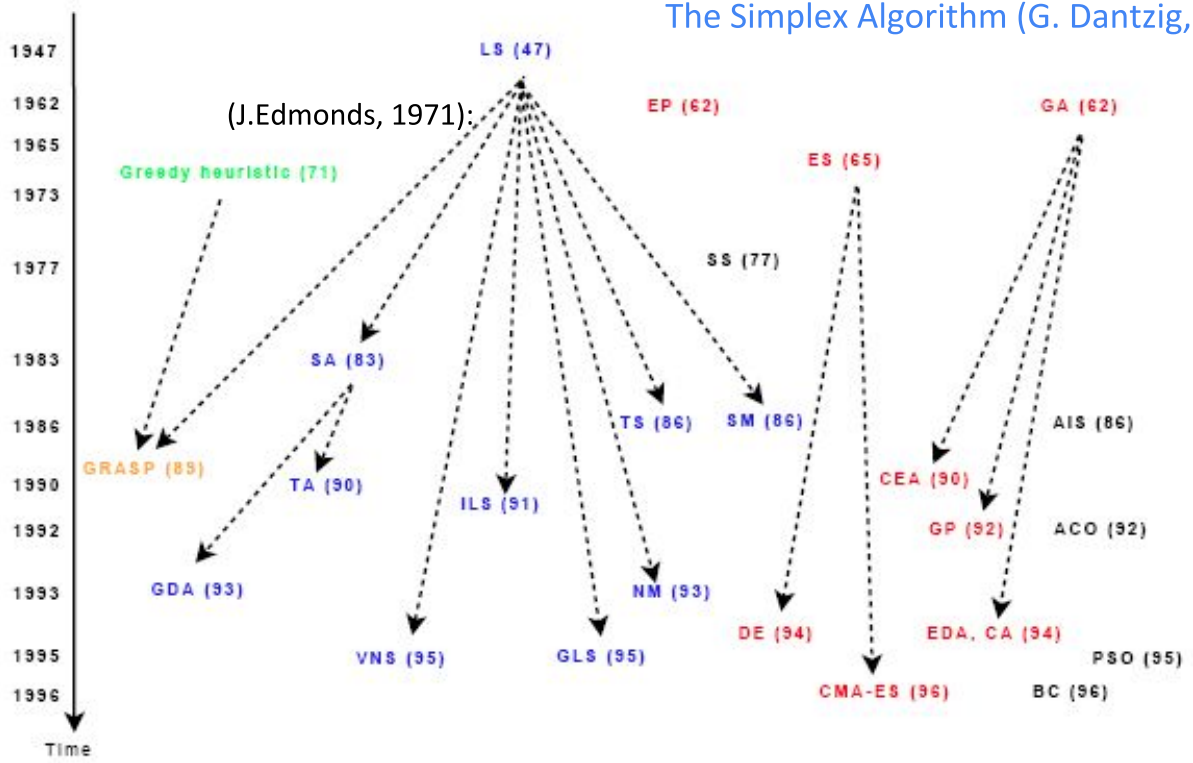
- Reject strategies: only feasible solutions are kept
- Penalising strategies: penalty functions
- Repairing strategies: repairing solutions
- Decoding strategies: only feasible solutions are generated
- Preserving strategies: specific representation and search operators which preserve the feasibility

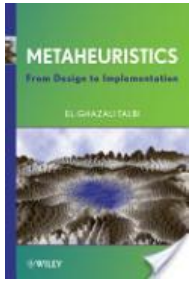
What is a metaheuristic?

- Extended variants of improvement heuristics
- General-purpose solvers, usually applicable to a large variety of problems
- Use two phases during search
  - Intensification (exploitation): focus the applications of operators on high-quality solutions
  - Diversification (exploration): systematically modifies existing solutions such as new areas of the search space are explored

Genealogy of metaheuristics

The Simplex Algorithm (G. Dantzig, 1947)





Metaheuristics: From Design to Implementation
By El-Ghazali Talbi (2009)

Towards effective search techniques

- Effective search techniques provide a mechanism to balance *exploration* and *exploitation*
  - exploiting the best solutions found so far
  - exploring the search space
- Extreme examples:
  - Hill-climbing methods *exploit* the best available solution for possible improvement but neglect exploring a large portion of the search space
  - Random search (points in the search space are sampled with equal probability) *explores* the search space thoroughly but misses exploiting promising regions.

Multiple restarts hill-climbing

Simple strategy to improve hill-climbing

Have an additional loop, repeat hill-climbing for a number of times from a different starting solution

```
procedure Multiple restart hill-climbing
begin
   s = randomly generated solution
   best = s
   repeat
      s = hill_climbing(s)
      if evaluation(s) is better than evaluation(best)
      best = s
      s = randomly generated solution
   until stopping-criteria satisfied
   return best
end
```

Iterated Local Search

Simple strategy to improve hill-climbing

Have an additional loop, repeat hill-climbing for a number of times from a *perturbed* solution

```
procedure Iterated local search
begin
  s = randomly generated solution
  s = hill-climbing(s)
  best = s
  repeat
     s = perturb(s)
     s = hill-climbing(s)
     if evaluation(s) is better than evaluation(best)
     best = s
  until stopping-criteria satisfied
  return best
end
```

Simulated annealing

- Key idea: provides a mechanism to escape local optima by allowing moves that worsen the objective function value
- *Annealing*: the physical process of heating up a solid and then cooling it down (slowly) until it crystallizes
  - candidate solutions ➔ states of physical system
  - objective function ➔ thermodynamic energy
  - globally optimal solutions ➔ ground states
  - parameter T ➔ physical temperature

Google Scholar citations: 43,512

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.

Simulated annealing – Algorithm

1. Start with a random solution *s*
2. Choose some “nearby” (a neighbour) solution *s’*
3. If the new solution is better (i.e. *f(s’) ≤ f(s)*) , take it as the current solution (= accept it)
4. If it is worse, accept it with a probability that depends on the deterioration *f(s)-f(s’)* and a global parameter *T* (the temperature)

Metropolis acceptance criterion

$$Paccept(T, s, s') = \begin{cases} 1 & \text{if } f(s') \leq f(s) \\ exp(\frac{f(s)-f(s')}{T}) & \text{otherwise} \end{cases}$$

Cooling schedule: mechanism for reducing the temperature

Summary of Metaheuristics

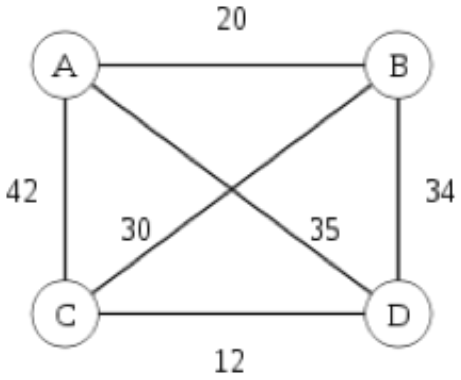
- Aim is to design search algorithms that can
  - escape local optima
  - balance exploration and exploitation
  - make the search independent from the initial configuration
- Successful single point search algorithms
  - Multiple restart hill-climbing
  - Iterated local search
  - Simulated annealing
  - Tabu search
  - Many others

A salesman must visit number of cities minimising the total cost of traveling

n cities and distance between each pair is given. We have to find a shortest route to visit each city exactly once and come back to the starting point.

Example solutions: permutation (ordering) of cities.

- s1= (A B C D), f(s1)= 20+30+12+35= 97
- s2= (A B D C), f(s2)= 20+34+12+42=108
- s3= (A C B D), f(s3)= 42+30+34+35= 141



Another combinatorial optimisation problem

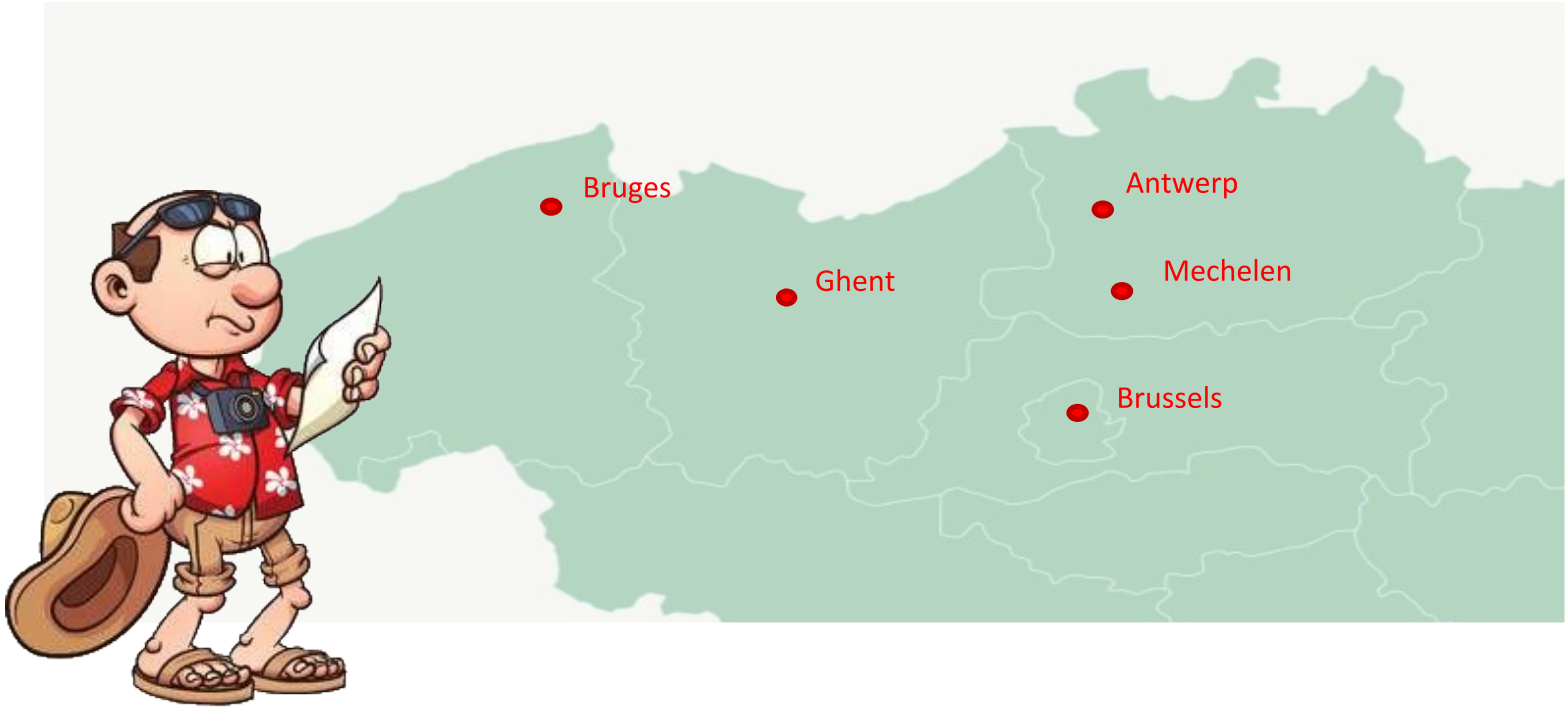
TRAVELING SALESMAN PROBLEM (TSP)

The Secret Rules of Modern Living Algorithms - math Prof Marcus Du Sautoy

TSP from minute 10:50

Travelling Salesman Problem

Tom flew into Brussels, and would like to visit four other cities in Belgium, before returning back to Brussels. In which order should Tom visit these cities, as to minimize his travel time?



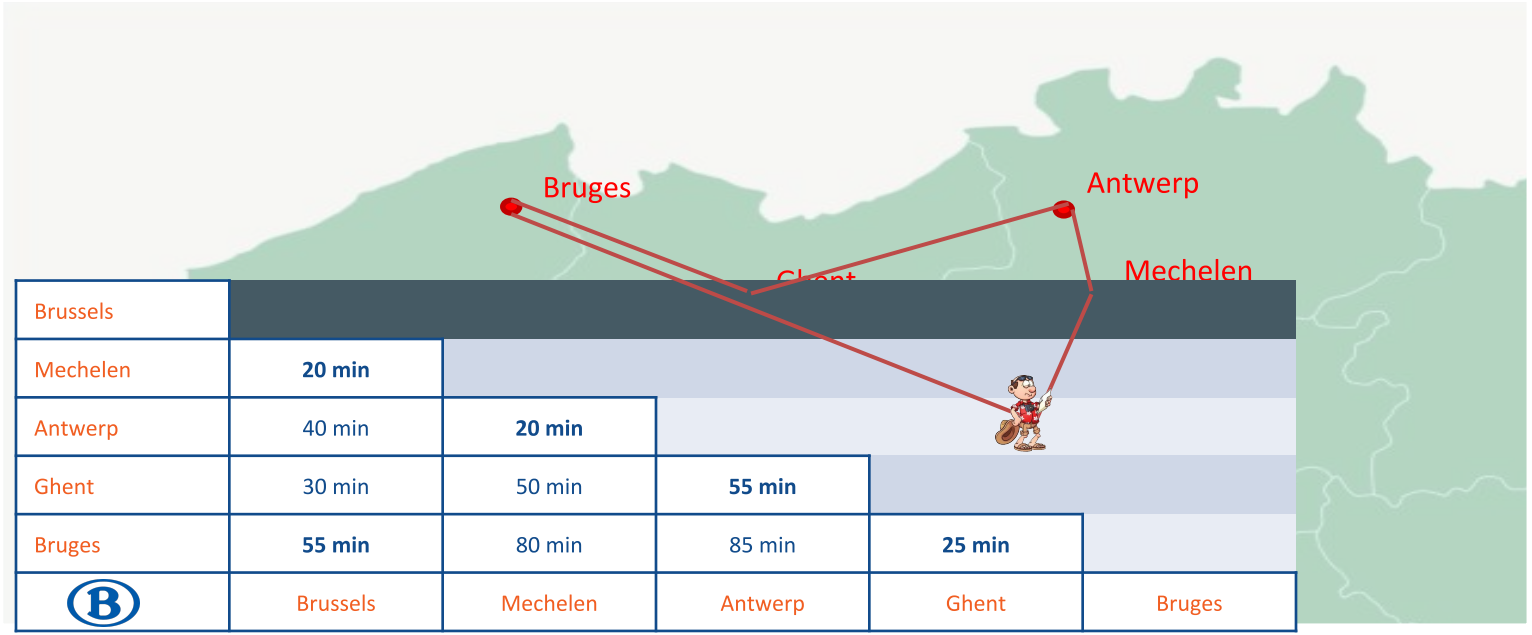
Traveling salesman problem

Tom is faced with a combinatorial optimization problem, known as the traveling salesman problem. Candidate solutions are possible tours Tom could follow, and the objective is to find the shortest one.

Brussels					
Mechelen	20 min				
Antwerp	40 min	20 min			
Ghent	30 min	50 min	55 min		
Bruges	55 min	80 min	85 min	25 min	
B	Brussels	Mechelen	Antwerp	Ghent	Bruges

Traveling salesman problem

For 5 cities, how many possible solutions there are?

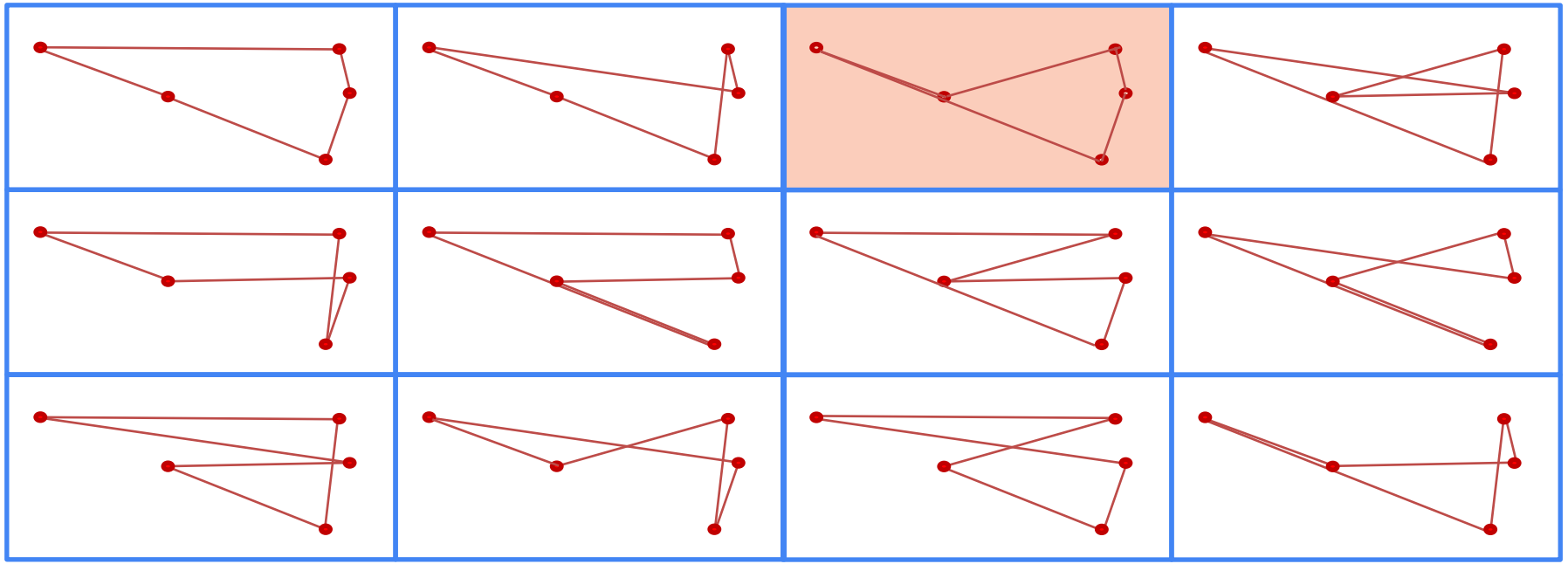


Brussels					
Mechelen	**20 min**				
Antwerp	40 min	**20 min**			
Ghent	30 min	50 min	**55 min**		
Bruges	**55 min**	80 min	85 min	**25 min**	
B	Brussels	Mechelen	Antwerp	Ghent	Bruges

Traveling salesman problem

Number of possible solutions in $(n-1)!/2$, where n is the number of cities

$4! = (4 * 3 * 2 * 1)/2 = 12$



For this small example: there are 12 possible tours, we could enumerate them all determine how long they take and return the shortest.

Combinatorial explosion...

Consider that Tom would be traveling through Europe and wishes to visit all 28 EU capitals.

In this case, there would be 5 billion, billion, billion possible tours...

billion billion billion

5,444,434,725,209,176,080,384,000,000 possible tours...

???