CS711008Z Algorithm Design and Analysis

Lecture 1. Introduction and some representative problems

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

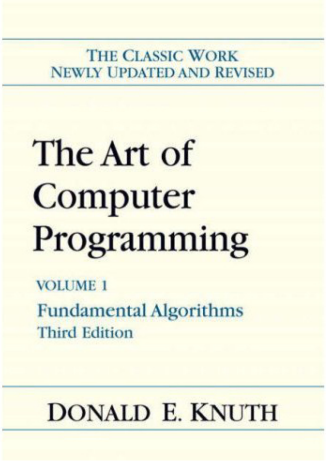
Figure 1: Muhammad ibn Musa al-Khwarizmi (C. 780—850), a Persian scholar, formerly Latinized as Algoritmi

- In the twelfth century, latin translations of his work on the Indian-Arabic numerals introduced the decimal positional number system to the western world.

Al-Khwarizmi's contributions

- Al-Khwarizmi's *The Compendious Book on Calculation by Completion and Balancing* presented the first systematic solution of **linear** and **quadratic equations** in Arabic.
- Two words:
  - **Algebra**: from Arabic "al-jabr" meaning "reunion of broken parts" — one of the two operations he used to solve equations
  - **Algorithm**: a step-by-step set of operations to get solution to a problem

Algorithm design: the art of computer programming



V. Vazirani said:

Our philosophy on the design and exposition of algorithms is nicely illustrated by the following analogy with an aspect of Michelangelos's art: A major part of his effort involved looking for interesting pieces of stone in the quarry and staring at them for long hours **to determine the form they naturally wanted to take***. The chisel work exposed, in a minimal manner, this form.*

V. Vazirani said: $cont'd$

By analogy, we would like to start with a clean, simply stated problem. Most of the algorithm design effort actually goes into **understanding the algorithmically relevant combinatorial structure of the problem**. *The algorithm exploits this structure in a minimal manner..... with emphasis on stating the structure offered by the problems, and keeping the algorithms minimal.*

(See extra slides.)

- DIVIDE-AND-CONQUER: Let's start from the "smallest" problem first, and investigate whether a large problem can **reduce to smaller subproblems**.
- IMPROVEMENT: Let's start from **an initial complete solution**, and try to improve it step by step.
- "INTELLIGENT" ENUMERATION: Consider an optimization problem. If the solution can be constructed step by step, we might enumerate **all possible complete solutions** by constructing a **partial solution tree**. Due to the huge size of the search tree, some techniques should be employed to prune it.

The first example: calculating the greatest common divisor (gcd)

The first problem: calculating **gcd**

Definition (gcd)

The greatest common divisor of two integers a and b, when at least one of them is not zero, is the largest positive integer that divides the numbers without a remainder.

- Example:
  - The divisors of 54 are: $1, 2, 3, 6, 9, 18, 27, 54$
  - The divisors of 24 are: $1, 2, 3, 4, 6, 8, 12, 24$
  - Thus, $\gcd(54, 24) = 6$.

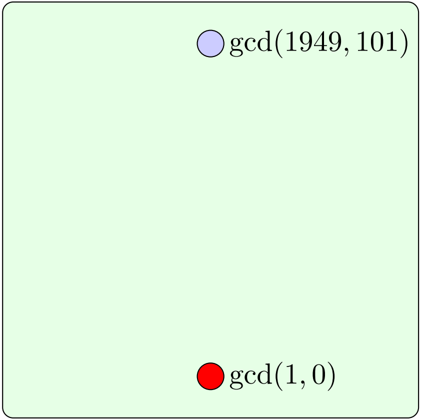
The first problem: calculating **gcd**

INPUT: two n-bits numbers a, and b ($a \geq b$)
OUTPUT: $\gcd(a, b)$

- Observation: the problem size can be measured by using n;
- Let's start from the "smallest" instance: $\gcd(1, 0) = 1$;
- But how to efficiently solve a "larger" instance, say $\gcd(1949, 101)$?

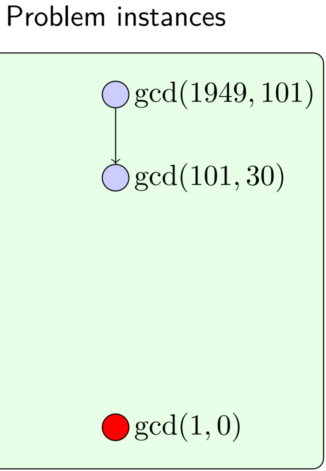
Euclidean algorithm

Problem instances



- Observation: a large problem can reduce to a smaller subproblem:
- $\gcd(1949, 101) = \gcd(101, 1949 \mod 101) = \gcd(101, 30)$ [1]

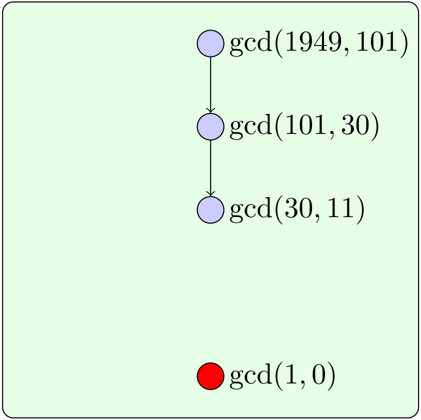
Strategy: reduce to "smaller" problems



- $\gcd(101, 30) = \gcd(30, 101 \mod 30) = \gcd(30, 11)$

Strategy: reduce to "smaller" problems

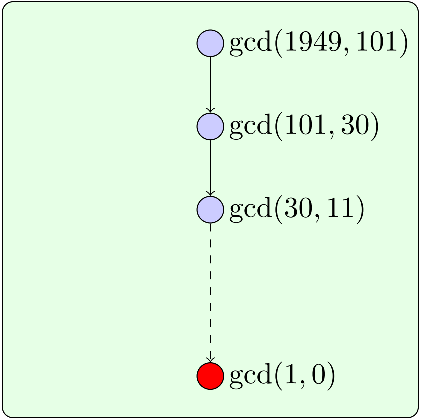
Problem instances



- $\gcd(30, 11) = \gcd(11, 30 \mod 11) = \gcd(11, 8)$

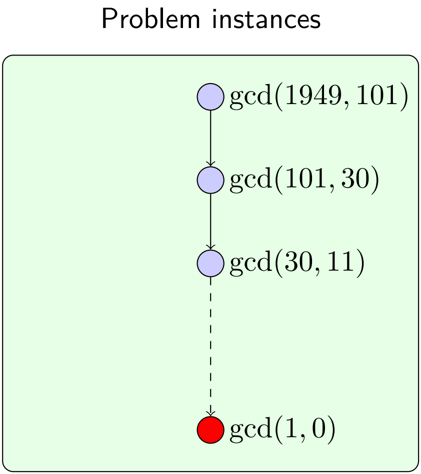
Strategy: reduce to "smaller" problems

Problem instances



- $\gcd(30, 11) = \gcd(11, 8) = \gcd(8, 3) = \gcd(3, 2) = \gcd(2, 1) = \gcd(1, 0) = 1$

Sub-instance relationship graph



- Node: subproblems
- Edge: reduction relationship

Euclid algorithm

function $\textsc{Euclid}(a, b)$

if $b = 0$ **then**
    return a;
end if
return $\textsc{Euclid}(b, a \mod b)$;

Time complexity analysis

Theorem

Suppose a is a n-bit integer. Euclid(a, b) ends in $O(n^3)$ time.

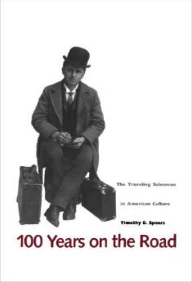
Proof.

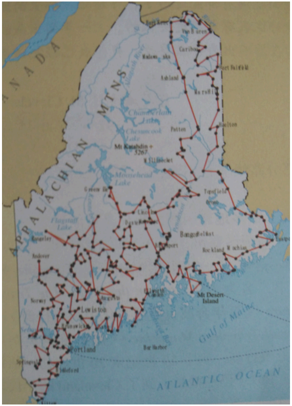
- There are at most $2n$ recursive calling.
  - Note that $a \mod b < \frac{a}{2}$.
  - After two rounds of recursive calling, both a and b shrink at least a half size.
- At each recursive calling, the $\mod$ operation costs $O(n^2)$ time.

$\square$

The second example: traveling salesman problem (TSP)

TSP: a concrete example





- In 1925, H. M. Cleveland, a salesman of the Page seed company, traveled 350 cities to gather order form.
- Of course, the shorter the total distance, the better.

How did they do? Pin and wire!

OFFICE APPLIANCES 201

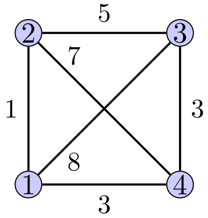
MAP CABINET Courtesy of Rand-McNally

202 SECRETARIAL STUDIES

MAP SHOWING ROUTING OF SALESMEN BY PINS AND CORDS
Courtesy of Rand-McNally

- Two pictures excerpted from *Secretarial Studies*, 1922.

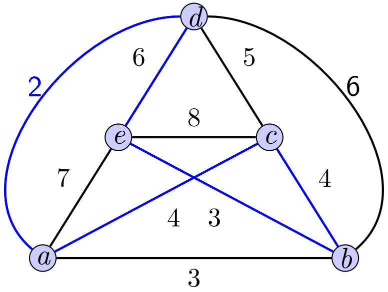
INPUT: n cities $V = \{1, 2, ..., n\}$, and a distance matrix D, where d_{ij} $(1 \leq i, j \leq n)$ denotes the distance between city i and j.
OUTPUT: the shortest tour that visits each city exactly once and returns to the origin city.

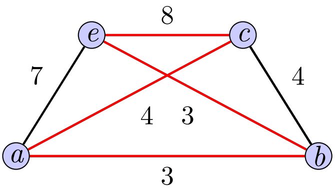


- Tour 1: $1 \to 2 \to 3 \to 4 \to 1$ (distance: 12)
- Tour 2: $1 \to 2 \to 4 \to 3 \to 1$ (distance: 19)
- Tour 3: $1 \to 3 \to 2 \to 4 \to 1$ (distance: 23)
- Tour 4: $1 \to 3 \to 4 \to 2 \to 1$ (distance: 19)
- Tour 5: $1 \to 4 \to 2 \to 3 \to 1$ (distance: 23)
- Tour 6: $1 \to 4 \to 3 \to 2 \to 1$ (distance: 12)

Trial 1: Divide and conquer

Decompose the original problem into subproblems

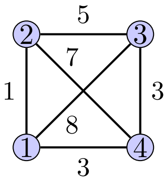




- Note that it is not easy to obtain the optimal solution to the original problem (e.g., tour in blue) through the optimal solution to subproblem (e.g., tour in red).

Consider a tightly-related problem

- Let's consider a tightly-related problem: calculating $M(s, S, x)$, the minimum distance, starting from city s, visiting each city in S once and exactly once, and ending at city x.
- It is easy to decompose this problem into subproblems, and the original problem could be easily solved if $M(s, S, x)$ were determined for all subset $S \subseteq V$ and $e \in V$.

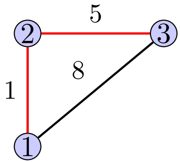


- For example, since there are 3 cases of the city from which we return to 1, the shortest tour can be calculated as:

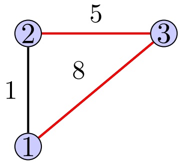
$$\begin{aligned} \min\{ \quad & d_{2,1} + M(1, \{3, 4\}, 2), \\ & d_{3,1} + M(1, \{2, 4\}, 3), \\ & d_{4,1} + M(1, \{2, 3\}, 4)\} \end{aligned}$$

Consider the smallest instance of $M(s, S, x)$

- It is trivial to calculate $M(s, S, x)$ when S consists of only 1 city.



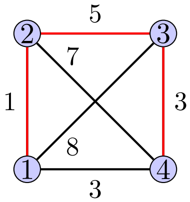
$M(1, \{2\}, 3) = d_{12} + d_{23}$

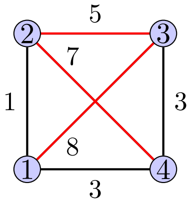


$M(1, \{3\}, 2) = d_{13} + d_{32}$

- But how to solve a larger problem, say $M(1, \{2, 3\}, 4)$?

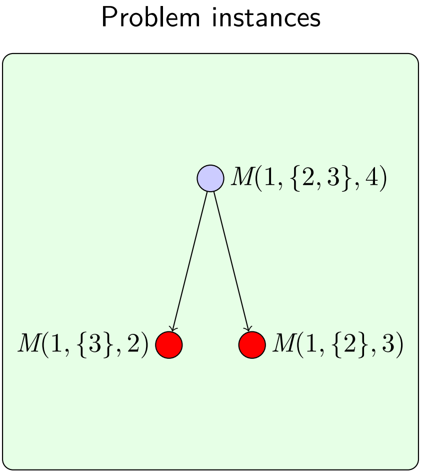
Divide a large problem into smaller problems





- $M(1, \{2, 3\}, 4) = \min\{d_{34} + M(1, \{2\}, 3), d_{24} + M(1, \{3\}, 2)\}$

Sub-instance relationship graph



- A large problem can be reduced into smaller subproblems.

Held-Karp algorithm [1962]

function $\mathrm{TSP}(D)$

return $\min_{e \in V, e \neq s} M(s, V - \{e\}, e) + d_{es}$;

function $\mathrm{M}(s, S, x)$

if $S = \{v\}$ **then**

  $\mathrm{M}(s, S, x) = d_{sv} + d_{vx}$;

  return $\mathrm{M}(s, S, x)$;

end if

return $\min_{i \in S,\ i \neq x} \mathrm{M}(s, S - \{i\}, i) + d_{xi}$;

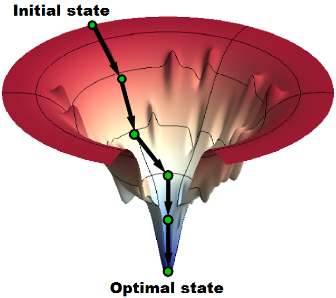
An example

S	x: b	c	d
$\{b\}$	-	7	6
$\{c\}$	8	-	12
$\{d\}$	10	15	-
$\{b, c\}$	-	-	15
$\{b, d\}$	-	14	-
$\{c, d\}$	15	-	-

- Time complexity: $O(2^n n^2)$.

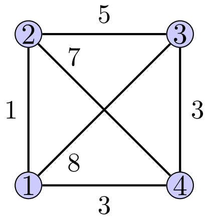
Trial 2: Improvement strategy

Solution space



- Landscape of solution space:
  - Node: a complete solution. Each node is associated with an objective function value.
  - Edge: if two nodes are neighboors, an edge is added to connect them. Here "neighbours" refers to two nodes with small difference.
- Improvement strategy: start from a rough complete solution, and try to improve it step by step.

IMPROVEMENT strategy



- Note that a **complete solution** can be expressed as a permutations of the n cities, e.g., $1, 2, 3, 4$.
- Let's start from **an initial complete solution**, and try to improve it;

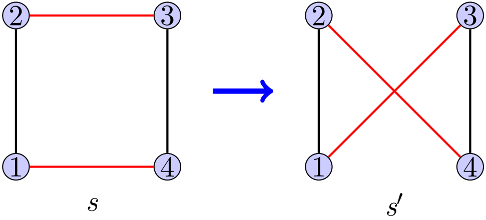
IMPROVEMENT strategy

```
function GenericImprovement(V, D)
  Let s be an initial tour;
  while TRUE do
      Select a new tour s' from the neighbourhood of s;
      if s' is shorter than s then
          s = s';
      end if
      if stopping(s) then
          return s;
      end if
  end while
```

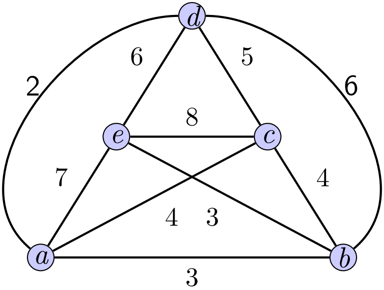
Here, **neighbourhood** is introduced to describe how to change an existing tour into a new one;

But how to define neighbourhood of a tour?

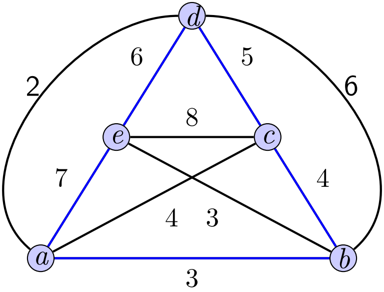
- 2-opt strategy: if s' and s differ at only two edges (Note: 1-opt is impossible)



An example

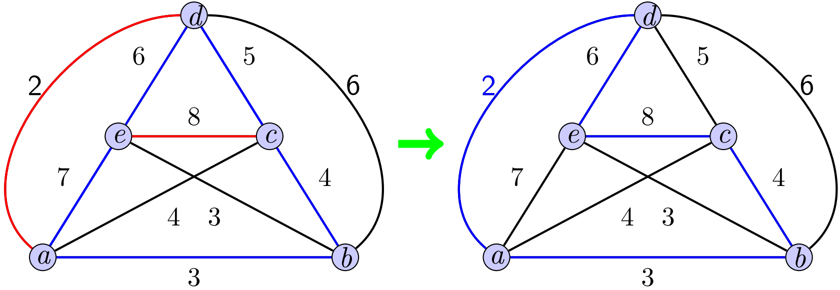


Step 1



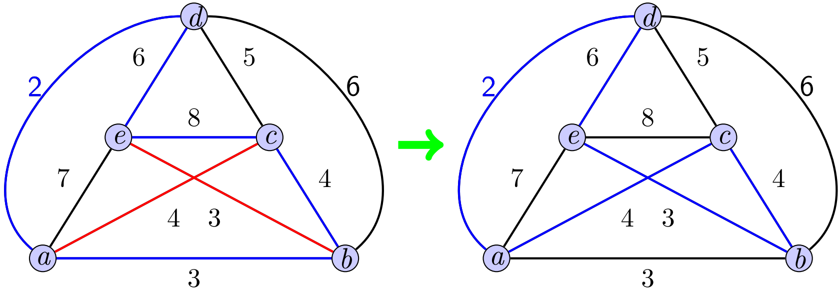
- Initial complete solution s: $a \to b \to c \to d \to e \to a$ (distance: 25)

Step 2: a 2-opt operation improves $s \Rightarrow s'$



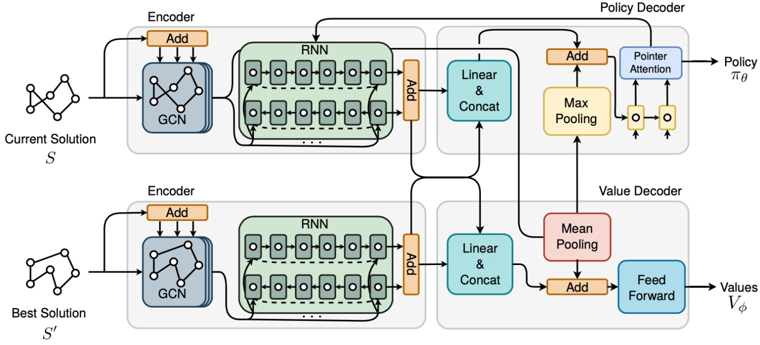
- Initial solution s: $a \to b \to c \to d \to e \to a$ (distance: 25)
- Improve from s to s': $a \to b \to c \to e \to d \to a$ (distance: 23)

Step 3: One more 2-opt operation improves $s' \Rightarrow s''$



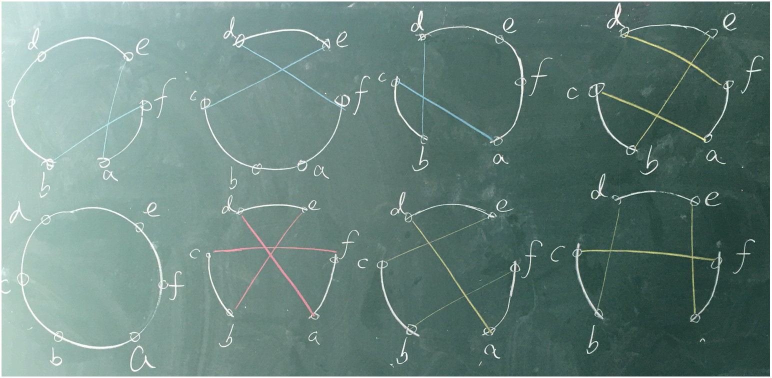
- A complete solution s': $a \to b \to c \to e \to d \to a$ (distance: 23)
- Improve s' to s'': $a \to c \to b \to e \to d \to a$ (distance: 19)
- Done! No 2-OPT can be found to improve further.

Recent development: Learning 2-OPT heuristics



- We can run 2-OPT on any edge-pair.
- How to select an appropriate edge-pair? Using NN and RL [Roberto2020].

Extension: 3-OPT and LKH technique

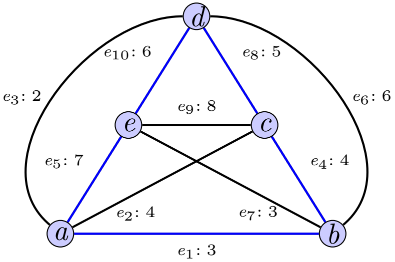


- Selecting three edge (a, b), (c, d), and (e, f), and applying $2 - OPT$ and $3 - OPT$ on them, we will have a total of 7 possible new tours.
- The LK-1 technique uses up to sequential $5 - OPT$ as well as non-sequential $4 - OPT$.

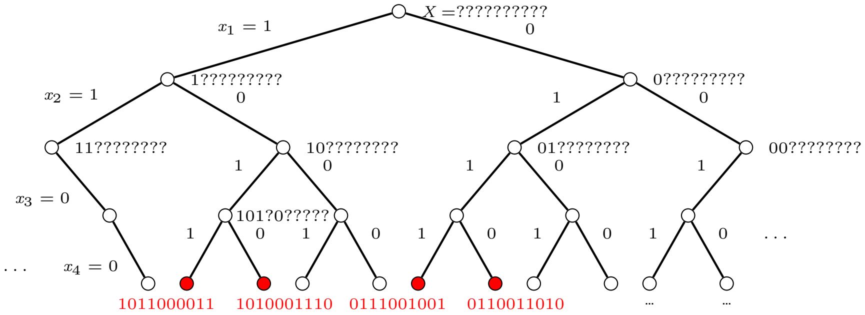
Trial 3: “Intelligent” enumeration strategy

Solution form

- Note that a complete solution can be expressed as a sequence of n edges. Given a certain order of the m edges, a complete solution can be represented as $X = [x_1, x_2, ..., x_m]$, where $x_i = 1$ if the edge i was used in the tour, and $x_i = 0$ otherwise.
- For example, the tour $a \to b \to c \to d \to e \to a$ can be represented as $X = [1, 0, 0, 1, 1, 0, 0, 1, 0, 1]$.
- As every solution is a combination of m items, we can enumerate all possible solutions.

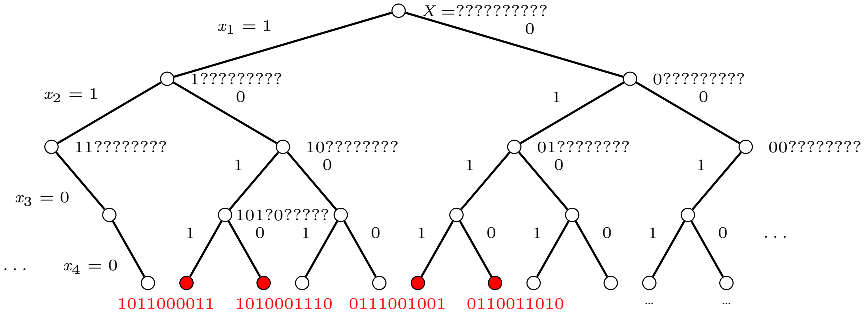


Organizing solutions into a partial solution tree



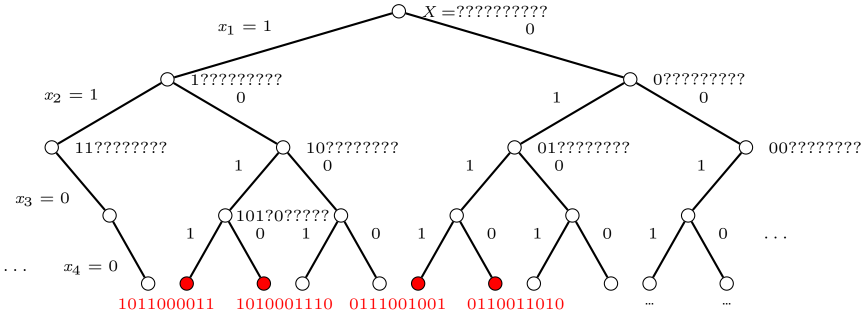
- Partial solution tree:
  - Leaf node: representing **a complete solution** associated with an objective function value, e.g., $X = 1011000011$ has an objective function value of 23.
  - Internal node: representing **a partial solution**, where only a subset of items (in the path from root to the node) are known, e.g., $X = 10????????$ refers to tours including e_1 but excluding e_2.

Two alternative views of a partial solution



- A subproblem: to determine the **unknown items**, e.g., $X = 10????????$ means to find the shortest tour including e_1 but excluding e_2.
- A collection of complete solutions: the leaf nodes of the subtree rooted at this partial solution, e.g., $X = 101?0?????$ represents two nodes in red 1011000011 and 1010001110.

Deduction rules for simplification



- Note that the following deduction rules were applied for simplification.
  - An edge (i, j) should be included in the tour if the removal of this edge leads to less than two edges incident to i or j, e.g., $x_8 = 1$ in $X = 1010001110$.
  - An edge (i, j) should be excluded from the tour if the addition of this edge leads to more than two edges incident to i or j, or leads to a small circuit, e.g., $x_5 = 0$ in $X = 101?0?????$.

Enumerate all complete solutions: Backtrack algorithm

function GENERICBACKTRACK

Let $A = \{X_0\}$. //Start with the root node $X_0 = ??...?$. Here, A denotes the active set of nodes that are unexplored

$best\_so\_far = \infty$;

while $A \neq NULL$ **do**

  Choose and remove a node X from A;

  Select an undetermined item in X, numerate this item, and thus **expand** X into nodes $X_1, X_2, ..., X_k$;

  for $i = 1$ to k **do**

    if X_i represents a complete solution **then**

      Update $best\_so\_far$ if X_i has better objective function value;

    else

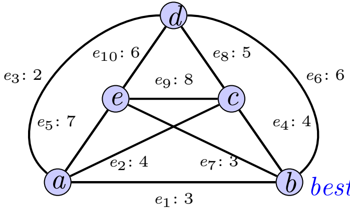
      Insert X_i into A; //X_i needs to be explored further;

    end if

  end for

end while

return $best\_so\_far$;



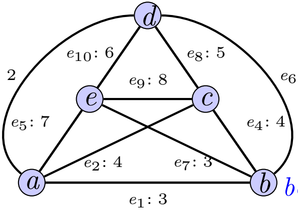
$A = \{X_0\}$
$best\_so\_far = +\infty$

Partial Solution Tree

$X = ??????????$
X_0

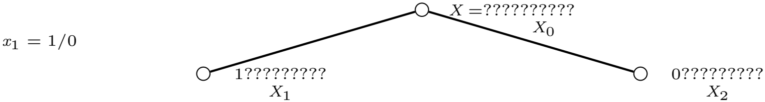
- Initially, the partial solution tree has only one root node, i.e., the original problem X_0. The value $best\_so\_far$ is set as $+\infty$.

Step 2



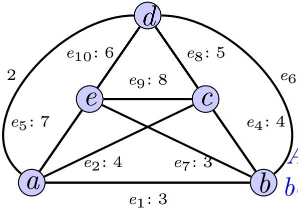
$A = \{X_1, X_2\}$
$best\_so\_far = +\infty$

Partial Solution Tree



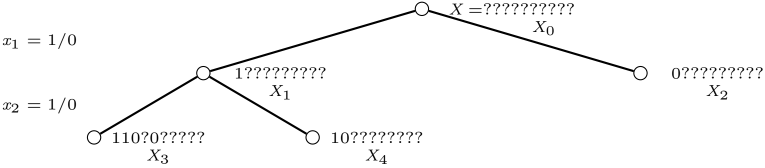
- X_0 is decomposed into two subproblems X_1 and X_2. We then expand X_1 at the next step.

Step 3



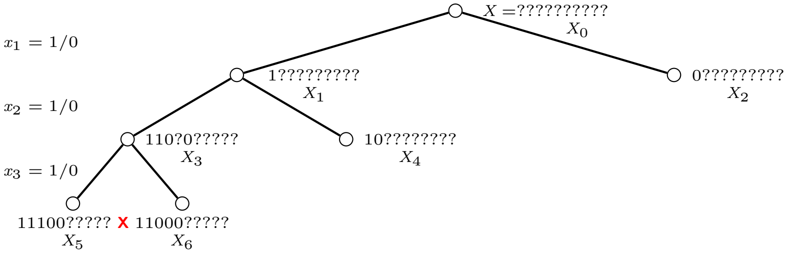
$A = \{X_2, X_3, X_4\}$
$best\_so\_far = +\infty$

Partial Solution Tree



- At this stage X_3 was selected for expansion.

Step 4

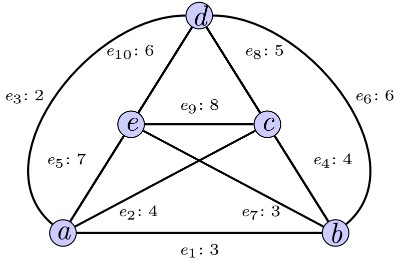


- In this way all possible complete solutions can be enumerated step by step, and it is unnecessary to store the whole tree in memory, thus reducing space requirement.
- Note that some internal nodes, say X_5, can be removed as the corresponding edges cannot form a valid tour.
- Theoretically speaking it will take exponential time to enumerate all possible complete solutions.
- Question: can we make the enumeration efficient?

Speeding up enumeration process by pruning branches

- Basic idea: to speed-up enumerating process, a feasible way is to prune branches at some internal nodes with "low quality". That is, **partial solutions can be exploited to exclude some complete solutions**. The representatives of this strategy include greedy method and branch-and-bound.
- Note that only complete solutions are associated with objective function value. Thus, how to estimate quality of a partial solution?
  - In greedy approaches, **heuristic functions** are used to estimate objective function value for a partial solution.
  - In branch-and-bound approaches, **lower bound functions** are used to calculate the lower bound of objective function value of a partial solution, i.e., the objective function value of all complete solutions represented by this partial solution.

Calculate lower bound for partial solution: an example



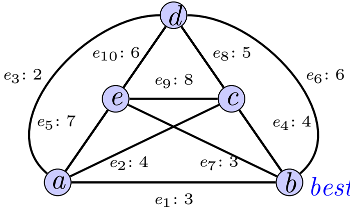
- For the partial solution $X =??????????$, we estimate the shortest tour as below:
  - For each city, we select the shortest two adjacent edges;
  - The sum of these $2n$ edges is less than or equal to 2 times the optimal tour;
  - Thus, we can give a lower bound as $\frac{1}{2}(5 + 6 + 8 + 7 + 9) = 17.5$.
- Similarly, the lower bound for the partial solution $X = 10????????$ was estimated as $\frac{1}{2}(5 + 6 + 9 + 7 + 9) = 18$.

“Intelligent” enumeration

function INTELLIGENTBACKTRACK

Let $A = \{X_0\}$. // Start with the root node $X_0 = ??...?$. Here A denotes the active set of nodes that are unexplored.
$best\_so\_far = \infty$;
while $A \neq NULL$ **do**
    Choose a node $X \in A$ **with lower bound less than** $best\_so\_far$, and remove X from A;
    Select an undetermined item in X, numerate this item, and thus **expand** X into nodes $X_1, X_2, ..., X_k$;
    for $i = 1$ to k **do**
        if X_i represents a complete solution **then**
            Update $best\_so\_far$ if X_i has better objective function value;
        else
            if **lowerbound**$(X_i) \leq best\_so\_far$ **then**
                Insert X_i into A; //X_i needs to be explored further
            end if
        end if
    end for
end while
return $best\_so\_far$;

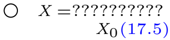
An example: Step 1



$A = \{X_0\}$

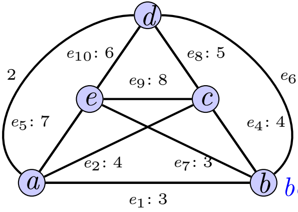
$best\_so\_far = +\infty$

Partial Solution Tree



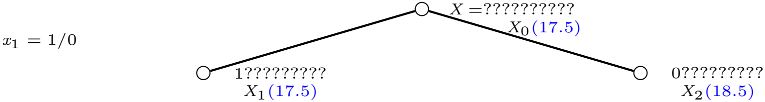
- Initially, the partial solution tree has only one root node, i.e., the original problem X_0 (lower bound: 17.5). The value $best\_so\_far$ is set as $+\infty$.

Step 2



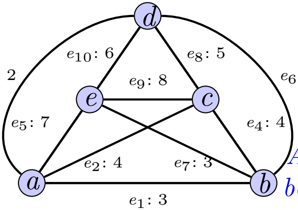
$A = \{X_1, X_2\}$
$best\_so\_far = +\infty$

Partial Solution Tree



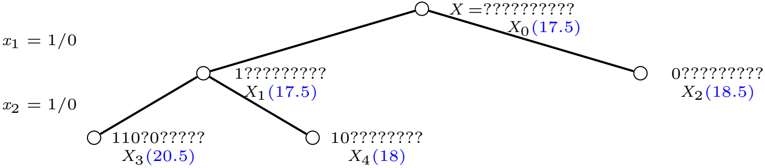
- X_0 is decomposed into two subproblems X_1 and X_2, and lower bounds of these two subproblems are estimated accordingly.
- As X_1 has a smaller lower bound than X_2, X_1 will be chosen to expand at the next step.
- Here, we adopt a rule to choose the subproblem with the smallest lower bound with the hope to find the optimal solution as soon as possible.

Step 3



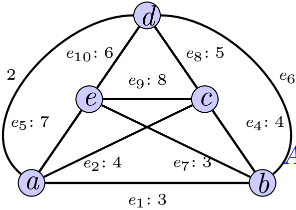
$A = \{X_2, X_3, X_4\}$
$best\_so\_far = +\infty$

Partial Solution Tree



- X_4 has the smallest lower bound; thus, it will be chosen for expansion.

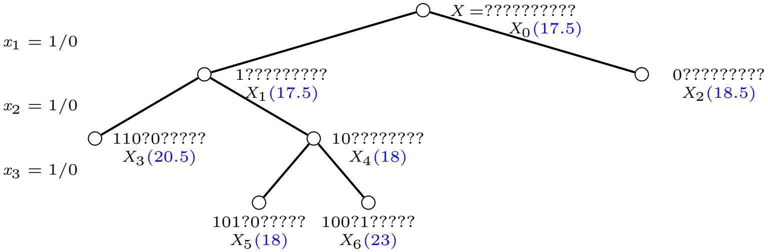
Step 4



$A = \{X_2, X_3, X_5, X_6\}$

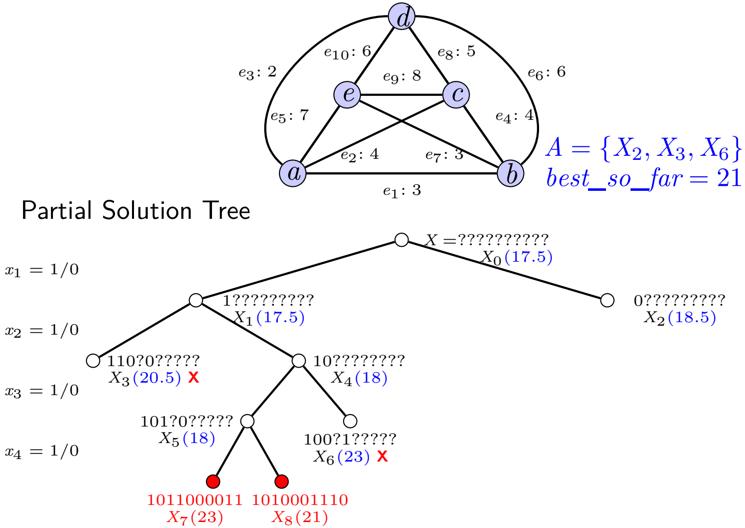
$best\_so\_far = +\infty$

Partial Solution Tree

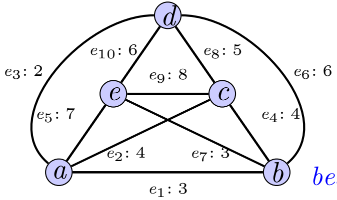


- X_5 has the smallest lower bound; thus, it will be chosen for expansion.

Step 5

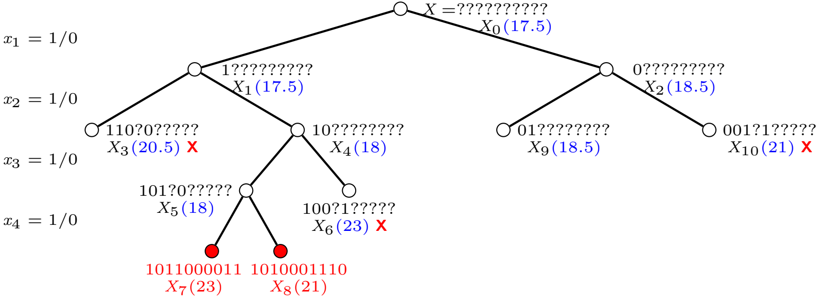


Two complete solutions, X_7 and X_8, are obtained, and $best\_so\_far$ is updated as 21. X_3 and X_6 should be removed, and X_2 will be chosen for expansion.



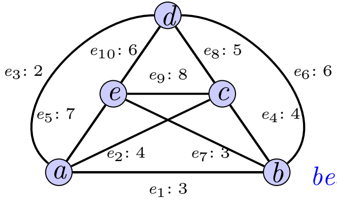
$A = \{X_9\}$
$best\_so\_far = 21$

Partial Solution Tree



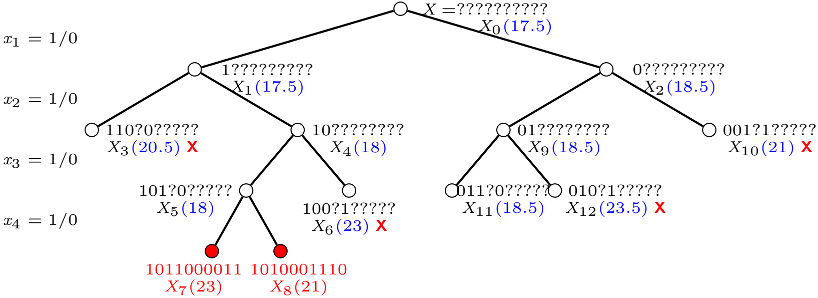
X_{10} should not be added into A, and X_9 is chosen for expansion.

Step 7



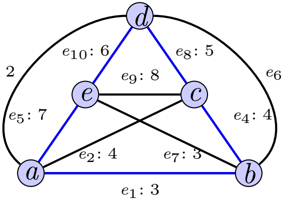
$A = \{X_{11}\}$
$best\_so\_far = 21$

Partial Solution Tree



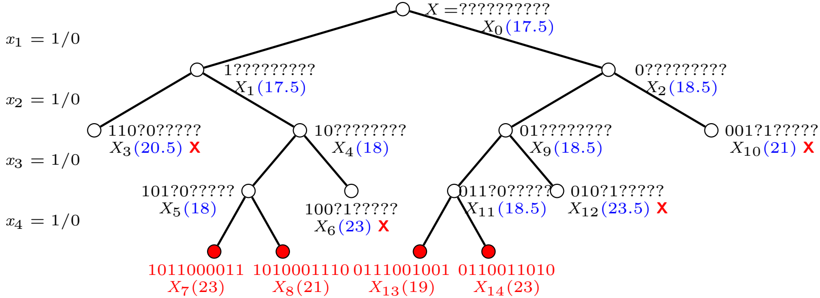
X_{12} should not be added into A, and X_{11} will be expanded.

Final step

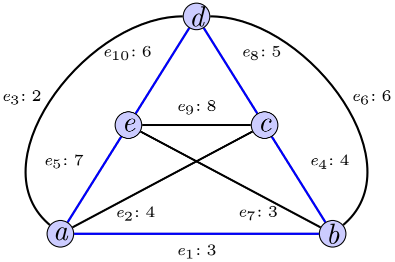


$A = \{\}$
$best\_so\_far = 19$

Partial Solution Tree



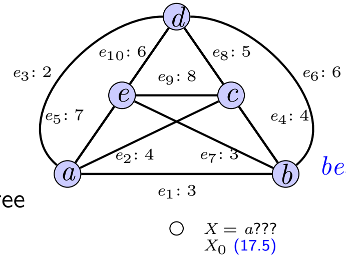
Note: the tree was pruned to contain only 15 nodes, making the algorithm efficient.



- Another solution form: Note that a tour can also be expressed as a sequence of n nodes, i.e., $X = [x_1, x_2, ..., x_{n-1}]$, where $x_i \in V$. Without loss of generality, we assume $x_1 = a$, and b should appear before c in the tour. For example, the tour in blue can be represented as $X = abcd$.
- As solutions have this form, we can enumerate them through building a partial solution tree together with pruning technique.

An example: Step 1

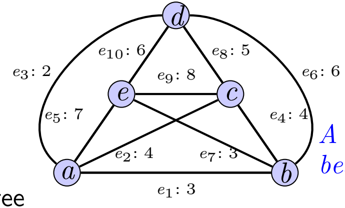
Partial Solution Tree



$A = \{X_0\}$
$best\_so\_far = \infty$

Initially, we have only one active sub-problem X_0 with lower bound of tour distance 17.5. X_0 will be expanded at the next step.

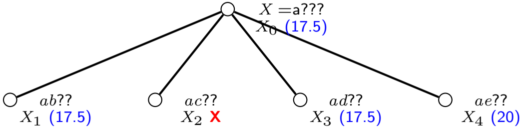
Step 2



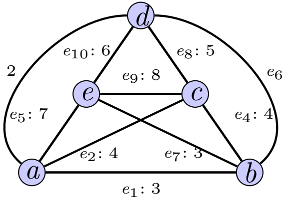
$A = \{X_1, X_3, X_4\}$
$best\_so\_far = \infty$

Partial Solution Tree

$x_2 = b/c/d/e$



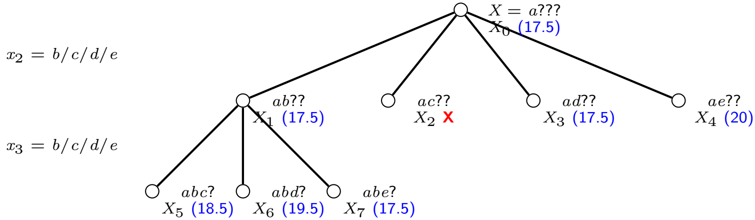
Note that the partial solution $X = ac??$ was abandoned as we assume b appears before c in the tour. X_1 will be expanded at the next step.



$A = \{X_3, X_4, X_5, X_6, X_7\}$

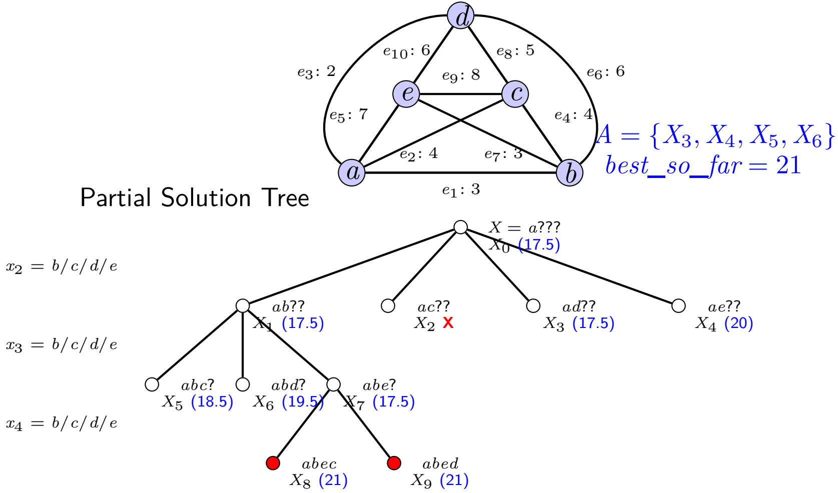
$best\_so\_far = \infty$

Partial Solution Tree



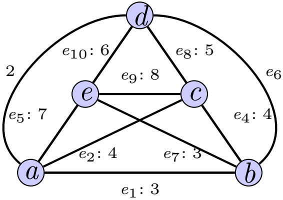
X_7 will be expanded at the next step.

Step 4



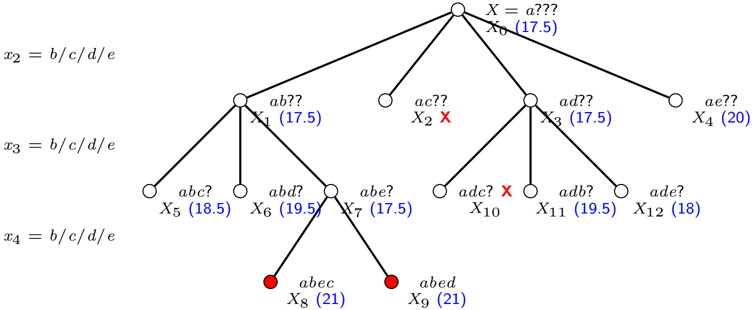
We obtained two complete solutions, and updated $best\_so\_far$ accordingly. X_3 will be expaned.

Step 5



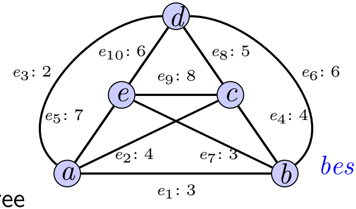
$A = \{X_4, X_5, X_6, X_{11}, X_{12}\}$
$best\_so\_far = 21$

Partial Solution Tree



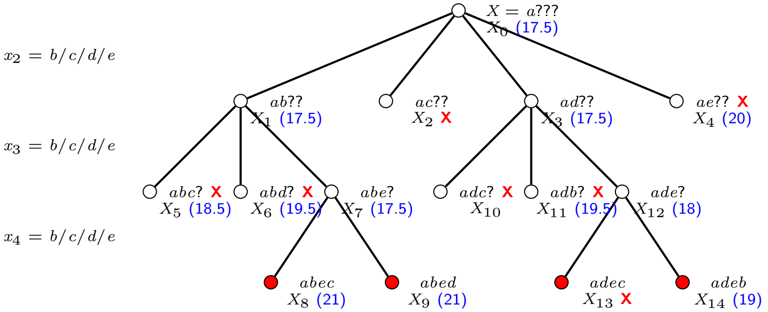
X_{12} will be expanded at the next step.

Final step



$A = \{\}$
$best\_so\_far = 19$

Partial Solution Tree



Note: the tree was pruned to contain only 15 nodes, making the algorithm efficient.

Time complexity and space complexity

Time complexity and space complexity

- Time (space) complexity of an algorithm quantifies the time (space) taken by the algorithm.
- Since the time costed by an algorithm grows with the size of the input, it is traditional to describe running time as a function of the input size.
  - **Input size**: The best notation of input size depends on the problem being studied.
    - For the TSP problem, the `number of cities in the input`.
    - For the MULTIPLICATION problem, the `total number of bits` needed to represent the input number is the best measure.

Running time: we are interested in its growth rate

- A straightforward way is to use the exact seconds that a program used. However, this measure highly depends on CPU, OS, compiler, etc.
- Several simplifications to ease analysis of algorithm:
  1. We simply use the number of primitive operations (rather than the exact seconds used) under the assumption that a primitive operation costs constant time. Thus the running time is $T(n) = an^2 + bn + c$ for some constants a, b, c.
  2. We consider only the leading term, i.e. an^2, since the lower order terms are relatively insignificant for large n.
  3. We also ignore the leading term's coefficient a since the it is less significant than the growth rate.
- Thus, we have $T(n) = an^2 + bn + c = O(n^2)$. Here, the letter O denotes order.

- Recall that big O notation is used to describe the error term in Taylor series, say:

$$e^x = 1 + x + \frac{x^2}{2} + O(x^3) \text{ as } x \to 0$$

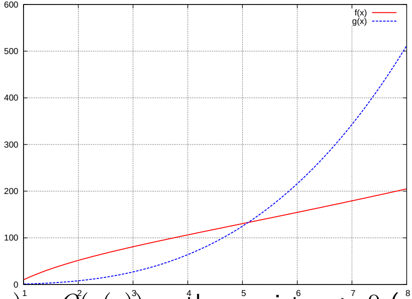


Figure 2: Example: $f(x) = O(g(x))$ as there exists $c > 0$ (e.g. $c = 1$) and $x_0 = 5$ such that $f(x) < cg(x)$ whenever $x > x_0$

Big Ω and Big Θ notations

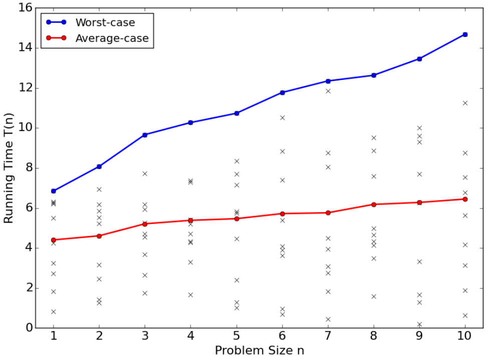
- In 1976 D.E. Knuth published a paper to justify his use of the Ω-symbol to describe a stronger property. Knuth wrote: "For all the applications I have seen so far in computer science, a stronger requirement [...] is much more appropriate".
- He defined

$$f(x) = \Omega(g(x)) \Leftrightarrow g(x) = O(f(x))$$

  with the comment: "Although I have changed Hardy and Littlewood's definition of Ω, I feel justified in doing so because their definition is by no means in wide use, and because there are other ways to say what they want to say in the comparatively rare cases when their definition applies".
- Big Θ notation is used to describe "$f(n)$ grows asymptotically as fast as $g(n)$".

$$f(x) = \Theta(g(x)) \Leftrightarrow g(x) = O(f(x)) \text{ and } f(x) = O(g(x)).$$

Worst case and average case



- Worst-case: the case that takes the longest time;
- Average-case: we need know the distribution of the instances;