

Design and Analysis of Algorithm

Backtrack (II)

- 1 Introduction to Branch and Bound
- 2 Knapsack Problem
- 3 Maximum Clique Problem (MCP)
- 4 Traveling Salesman Problem

Outline

- 1 Introduction to Branch and Bound
- 2 Knapsack Problem
- 3 Maximum Clique Problem (MCP)
- 4 Traveling Salesman Problem

Combinatorial Optimization

Combinatorial Optimization. finding an optimal solution x from a finite set of feasible/candidate solutions S .

- Constraint: $P(x) = 1 \iff x \in S$
- Optimized function $f(x) \rightarrow$ define optimal solution
typical optimized function aims to maximize or minimize

Example of Knapsack

- $P(x) : 2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$
- $f(x) : \max\{x_1 + 3x_2 + 5x_3 + 9x_4\}$

W.L.O.G always assume that maximization of $f(x)$ is desired

- since one can find the minimum value of $f(x)$ by finding the maximum of $g(x) = -f(x)$

Extensively studied in operations research, applied mathematics and theoretical computer science.

- traveling salesman problem (TSP), minimum spanning tree problem (MST), knapsack problem

Motivation

Generally, combinatorial optimization problem can be solved via enumeration of candidate solutions and testing them all

- enumeration can be done by **brute-force** searching the state space tree
 - leaf node \Leftrightarrow candidate solution

For \mathcal{NP} -hard problem, the state space could be exponentially large.

Can we improve on the performance of brute-force search of state space tree?

Branch-and-Bound Method

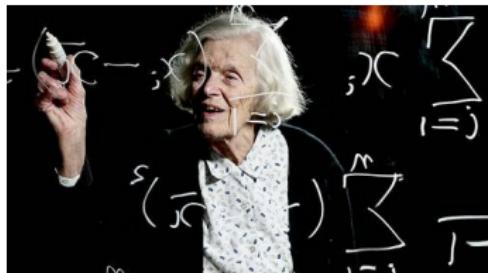
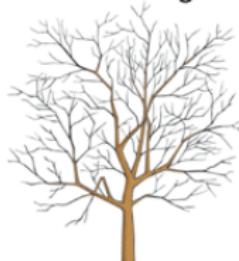


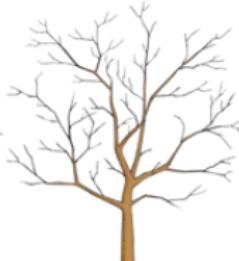
Figure: 1960, British: Ailsa Land and Alison Harcourt

the most commonly used technique for solving \mathcal{NP} -hard optimization problems

A Look at Pruning



GOOD



NOT GOOD



Key Elements of Branch-and-Bound

A B&B algorithm operates according to two principles:

- **Branching.** recursively split the search space into smaller spaces, then try to find maximal $f(x)$ on these smaller spaces
 - **Bounding.** keep track of a **bound value**, compute an **upper bound** of $f(x)$ in the smaller space, and use this **bound value** and the **upper bound** to “prune” the search space, eliminate candidate solutions that will not contain an optimal solution
-

Key points of good “pruning”

- the setting of **bound value** (typically natural)
- the calculation of **upper bound** (relatively tricky)

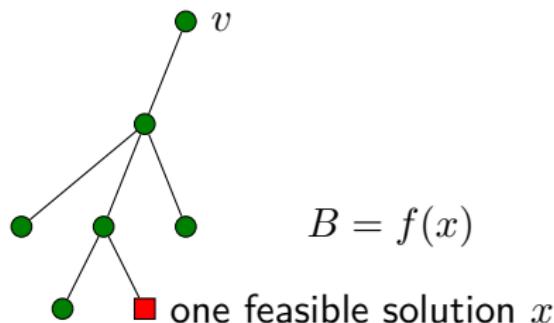
Bound Value

Meaning: maximal optimized function value of current feasible solutions

Initial Value: 0 for maximize problem and ∞ for minimize problem

Update

- when find the first solution
- when find a better solution

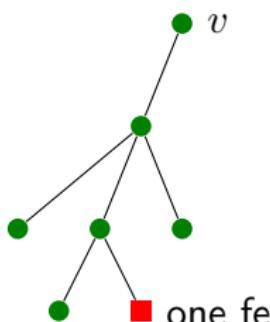


Estimate/Bound Function

Input: node of tree, say, v

Output: an upper bound of maximum value of all feasible solutions in the subtree with input node as root

Soundness: let v' be a child node of v



$$E(v) \geq E(v')$$

the feasible solution set shrinks
⇒ the upper bound is getting smaller
soundness ⇒ will not miss solution

The choice of estimate function is not unique, one need to make a trade-off between

- computation cost vs. accuracy

A Metaphor of Bound Function (represent the most optimistic estimation)



Backtracking and Pruning

When navigating to a node v , the algorithm will stop branching and backtrack to parent node if one of the two following cases occurs:

- ① the set of feasible solutions $A(v)$ is empty
 - no leaf node in the subtree satisfies the constraint predicate (same as naive backtracking with default constraint)
- ② the estimate function value is less than current bound value

$$E(v) < B$$

- simply prune this subtree and backtrack

Outline

- 1 Introduction to Branch and Bound
- 2 Knapsack Problem
- 3 Maximum Clique Problem (MCP)
- 4 Traveling Salesman Problem

Example from Knapsack Problem with Repetition

Knapsack problem instance (weight limit $W = 10$)

label	weight	value
1	2	1
2	3	3
3	4	5
4	7	9

Constrained predicate P :

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$

Optimized function f : $\max\{x_1 + 3x_2 + 5x_3 + 9x_4\}$

Choice of Bound Function

For each node $v = (x_1, \dots, x_k)$, compute the upper bound of optimized function value of feasible solutions in the subtree

- Preprocessing: sort v_i/w_i via a decreasing order, $i \in [n]$

$$\underline{E(v) = K(v) + \Delta(v)}$$

$K(v)$: the value already loaded in the knapsack

$\Delta(v)$: the maximum value that can be further loaded

Computation of $\Delta(v = (x_1, \dots, x_k))$:

- $\Delta(v) = \text{remaining weight} \times v_{k+1}/w_{k+1}$

Obviously, there exists more fine-grained method of computing $E(v)$, e.g., finding the first loadable item and then computing $E(v)$ accordingly.

Knapsack Instance

$$\max\{x_1 + 3x_2 + 5x_3 + 9x_4\}$$

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$

Re-arrange the label such that

$$\frac{v_i}{w_i} \geq \frac{v_{i+1}}{w_{i+1}}$$

After rearrangement

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$

Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$

estimate value
current weight

branch

backtrack by Dominal property

update bound value

bound and prune

Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$

estimate value
current weight

$$\frac{10 \cdot 9 / 7}{0}$$

branch

backtrack by Dominal property

update bound value

bound and prune

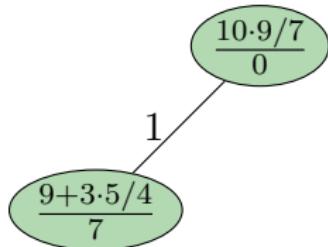
Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$

estimate value
current weight

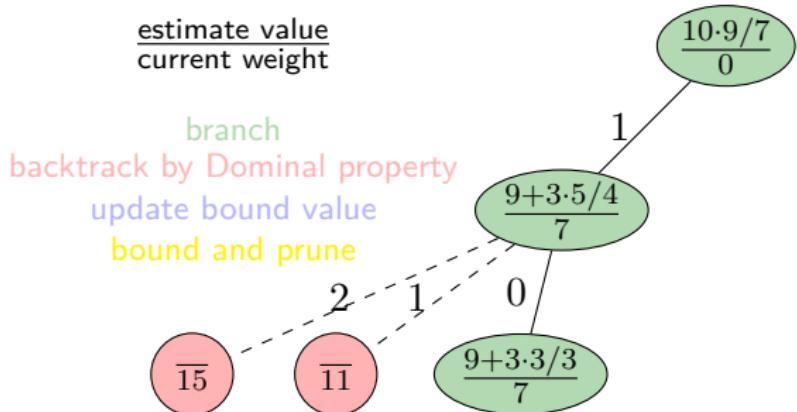
branch
backtrack by Dominal property
update bound value
bound and prune



Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

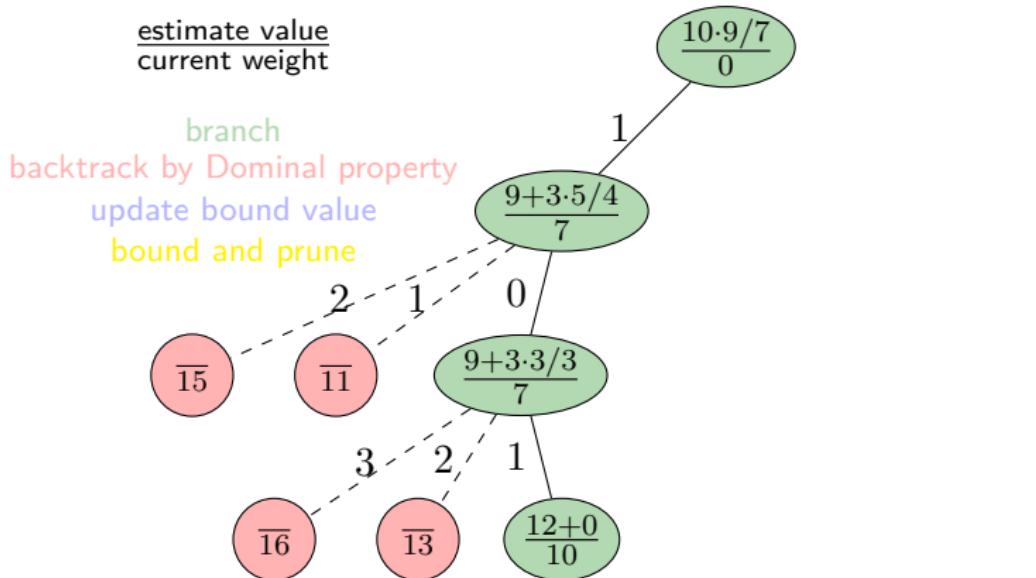
$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$



Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

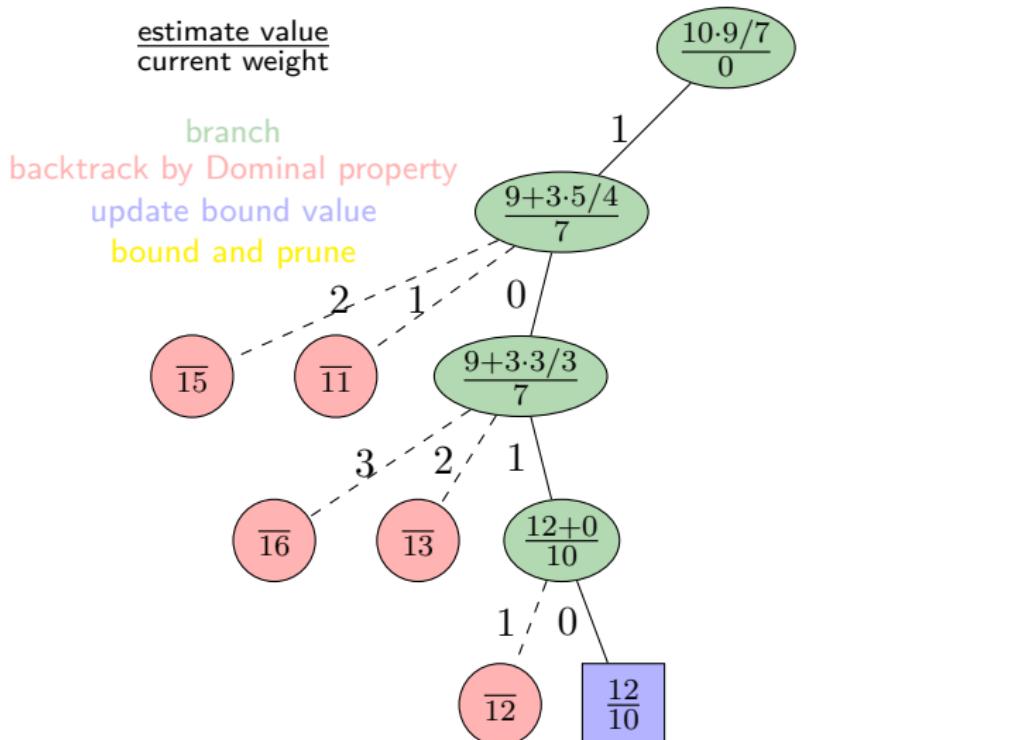
$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$



Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

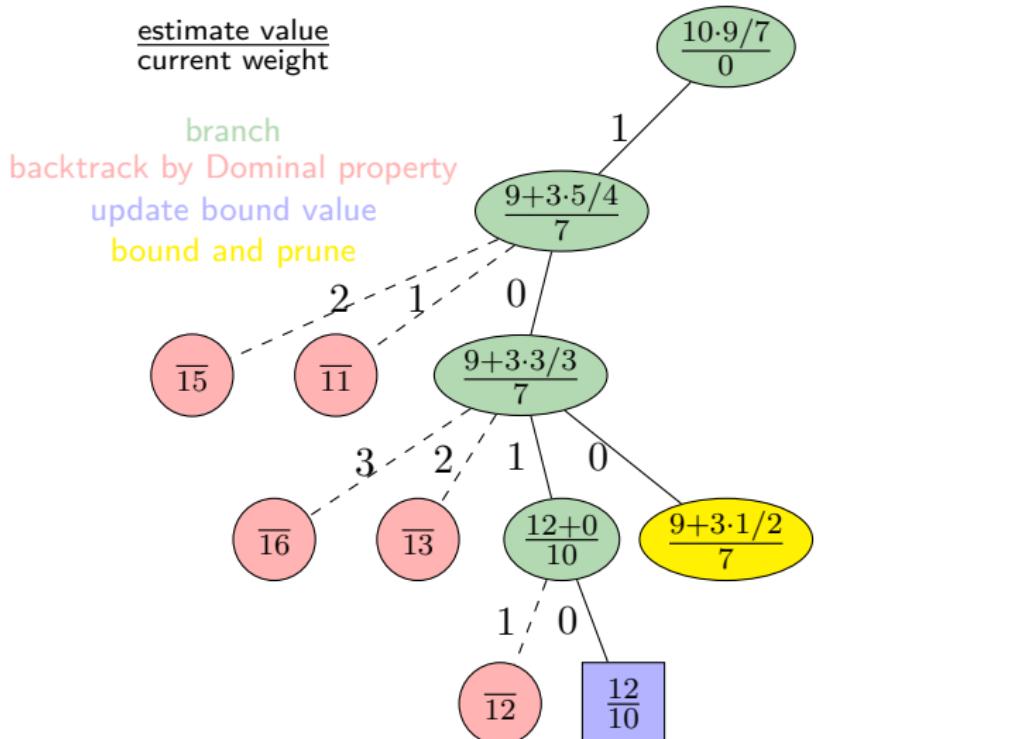
$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$



Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

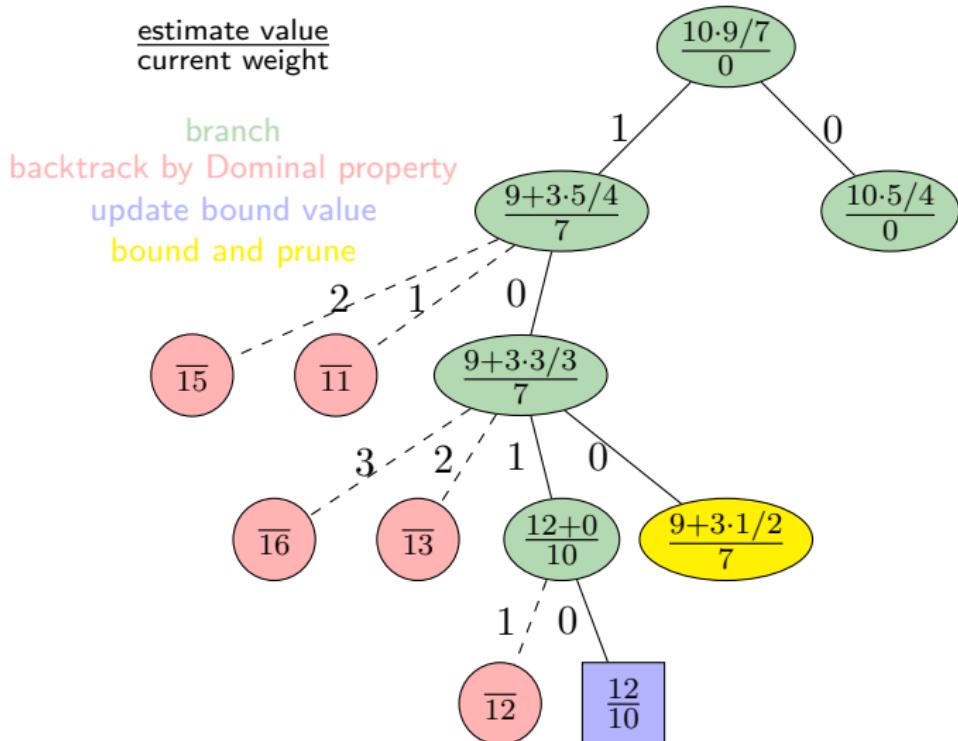
$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$



Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

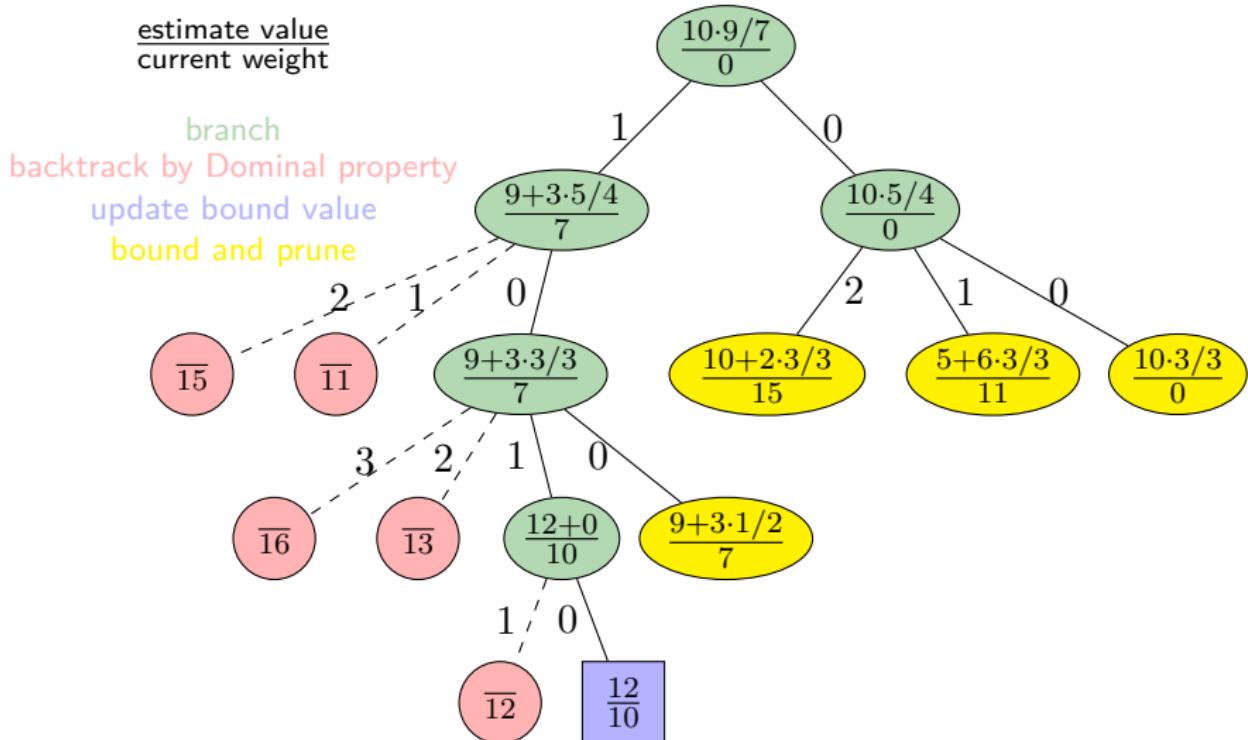
$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$



Branch-and-Bound Example

$$\max\{9x_1 + 5x_2 + 3x_3 + x_4\}$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10, x_i \in \mathbb{N}, i \in [4]$$



Thinking and Summary

- Q. Is the preprocessing step necessary?
- A. No. But it can speed up the computation of bound function.
- Q. Other possible choice of bound function?
- A. Yes. But make sure it is easy to compute, hit a **sweet balance** between the cost and gain of pruning
-

Branch-and-Bound method \leadsto Combinatorial Optimization

- bound value setting and updating
- estimate function (represent the optimistic estimation) \Rightarrow guarantee that pruning will not miss solution
- pruning: compare bound value and estimate function value

Outline

- 1 Introduction to Branch and Bound
- 2 Knapsack Problem
- 3 Maximum Clique Problem (MCP)
- 4 Traveling Salesman Problem

Concepts of Graph

Let $G = (V, E)$ be an undirected graph

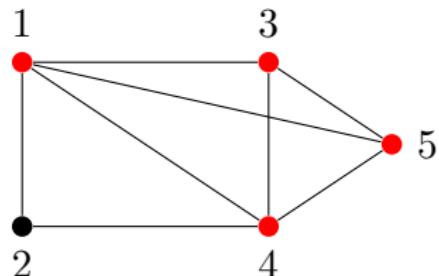
Subgraph: $G' = (V', E')$, where $V' \subseteq V$, $E' \subseteq E$

Cograph: $\overline{G} = (V, \overline{E})$, where \overline{E} is the co-set of E regarding the complete graph over V

Clique: A complete subgraph of G

Maximum Clique: A clique with the largest possible number of vertices.

- MCP is a classical combinatorial optimization problem in graph theory.



maximum clique = {1, 3, 4, 5}

Independent Set and Clique (独立集与团)

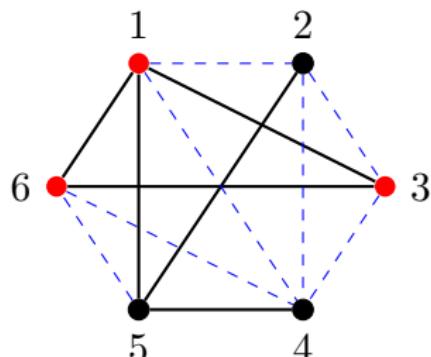
Let $G = (V, E)$ be an undirected graph

Independent Set. A subset U of V , s.t. $\forall u, v \in U, (u, v) \notin E$

Maximum Independent Set. An independent set of largest possible size for G .

Claim: U is the maximum clique of G if and only if U is the maximum independent set of \bar{G} .

independent set \leftrightarrow clique in co-graph



$\{1, 3, 6\}$
maximum clique of G
maximum independent set of \bar{G}

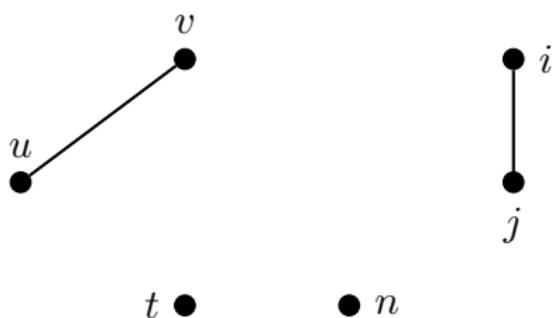
Applications of Maximum Clique

Numerous applications of MCP

- coding, cluster analysis, computer vision, economics, mobile communication, VLSI design

Example from coding. Noisy in the channel may disturb code transmission. Consider confusion graph $G = (V, E)$, V is a finite set of symbols

$(u, v) \in E$ or $E(u, v) = 1 \Leftrightarrow u$ and v are likely confused

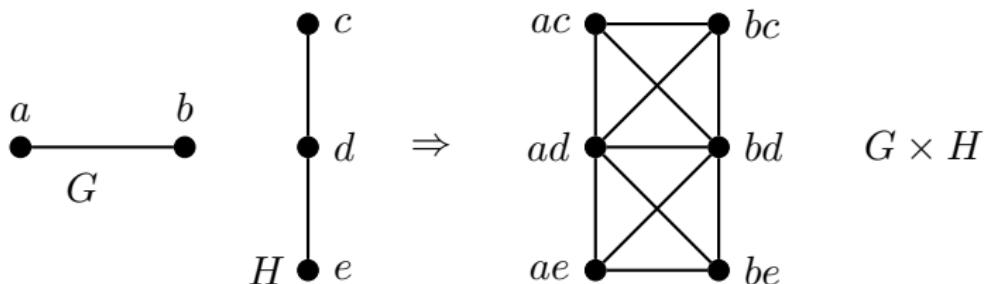


Coding Design

In coding design, we usually use a string to encode a symbol.

Confusion of codeword. We say two strings xy and uv are likely to be confused if and only if

$$(E(x, u) = 1 \wedge E(y, v) = 1) \vee \\ (x = u \wedge E(y, v) = 1) \vee (E(x, u) = 1 \wedge y = v)$$



Vetics in $G \times H$ are candidate codewords

- two codewords are confused if there is an edge between them

To reduce noisy disturb, we need to find MIS in $G \times H$.

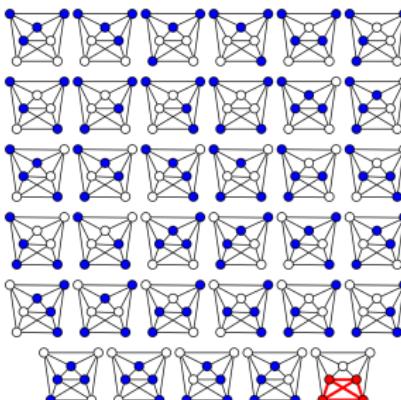
Maximum Clique Problem (最大团)

Problem. Given an undirected graph $G = (V, E)$, where $V = \{1, \dots, n\}$, find its maximum clique.

Solution. An n -dimension vector $(x_1, x_2, \dots, x_n) \in \{0, 1\}^n$, $x_k = 1$ if and only if k is in the maximum clique of G .

Brute Force Algorithm. For every subset of V , check if it forms a clique, i.e., a complete subgraph.

subsets of V is $2^n \rightsquigarrow$ exponential time complexity $O(n^2) \cdot 2^n$



Branch-and-Bound Method

Search tree: Subset tree (a binary tree: the path from leaf node to root determines a subset)

Node (x_1, x_2, \dots, x_k) : have checked nodes $1, 2, \dots, k$, $x_i = 1$ denotes i belongs to the current clique, $i \in [k]$

Constraint. $x_{k+1} = 1$ if and only if it connects to all the nodes in the current clique

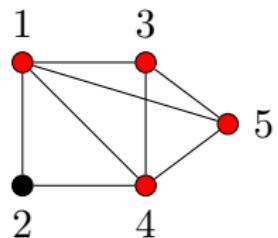
Bound value: # vertices in the current maximum clique

Estimate function: the largest number of vertices that current clique may expand to: $E(v) = C(v) + (n - k)$.

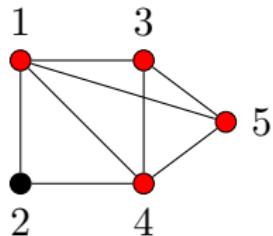
- $C(v)$: # of vertices in the current clique (initial value is 0)
- k : the depth of v

E is simple but too coarse \leadsto worse-case complexity is $O(n2^n)$, asymptotically same as the brute-force algorithm

Demo of Search (the perfect binary tree)



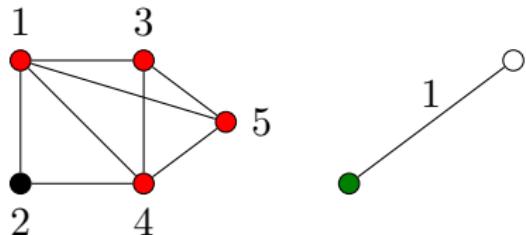
Demo of Search (the perfect binary tree)



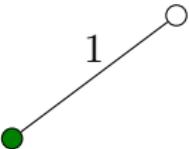
○

initial value $B = 0$

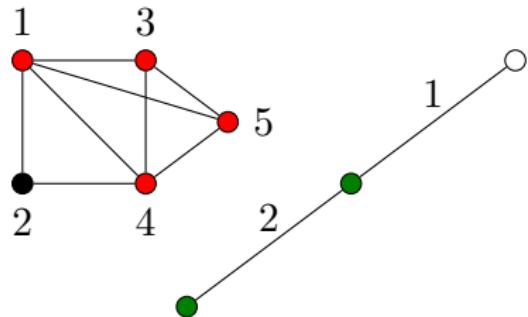
Demo of Search (the perfect binary tree)



initial value $B = 0$

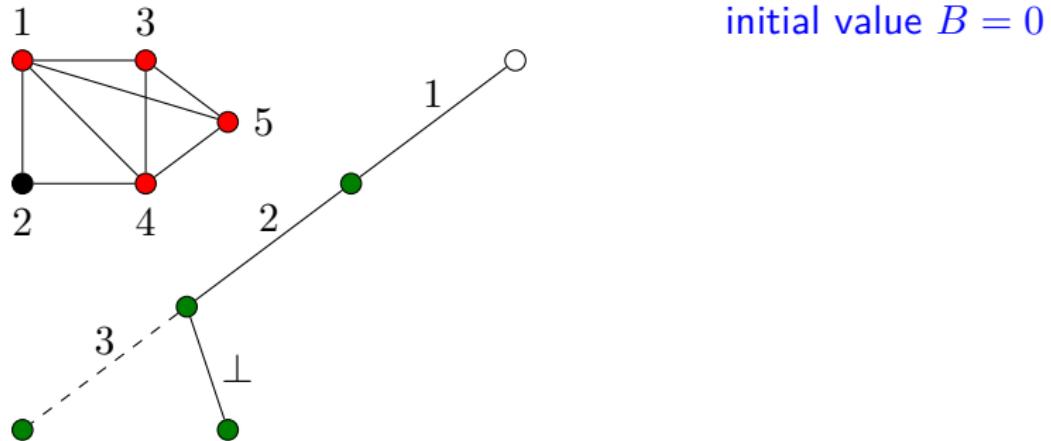


Demo of Search (the perfect binary tree)

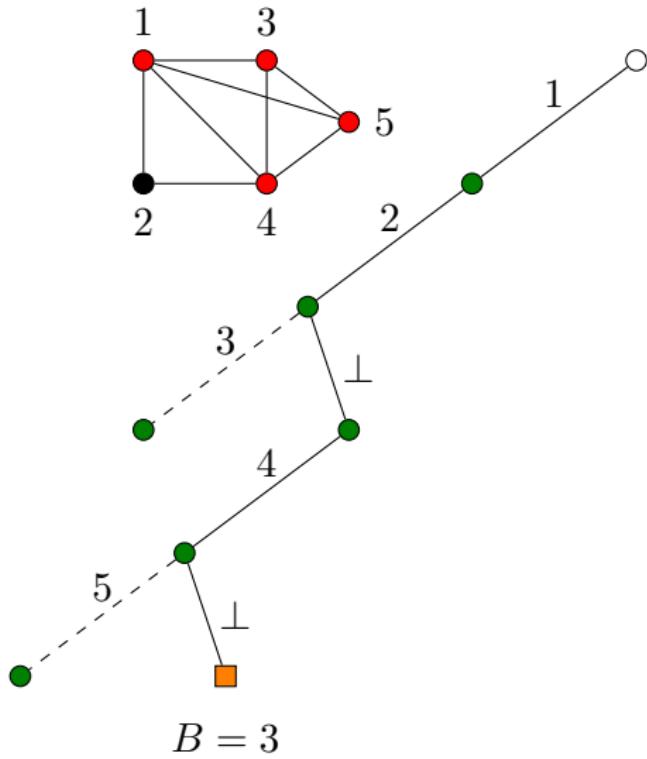


initial value $B = 0$

Demo of Search (the perfect binary tree)



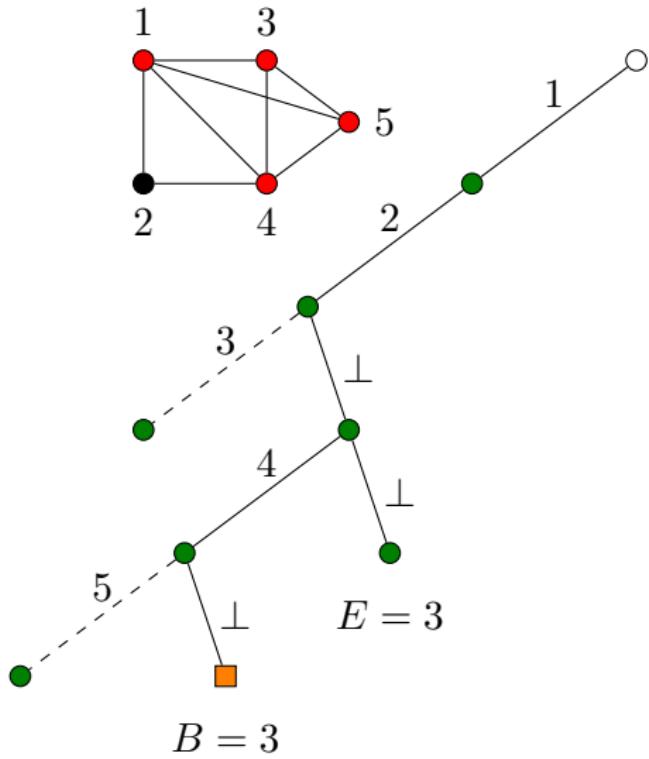
Demo of Search (the perfect binary tree)



initial value $B = 0$

$\text{MC} = \{1, 2, 4\}, B = 3$

Demo of Search (the perfect binary tree)

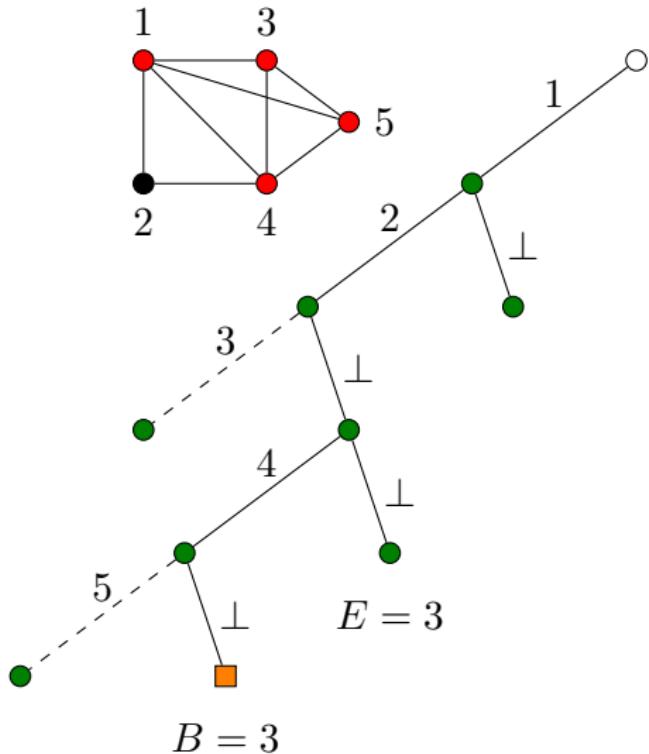


initial value $B = 0$

$MC = \{1, 2, 4\}, B = 3$

$E = 3 \leq B$, backtrack

Demo of Search (the perfect binary tree)

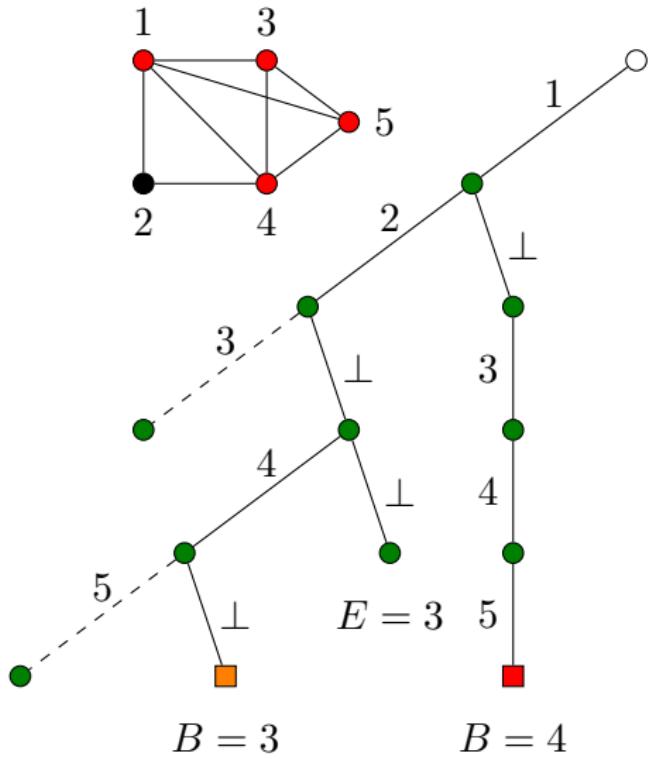


initial value $B = 0$

$MC = \{1, 2, 4\}, B = 3$

$E = 3 \leq B$, backtrack

Demo of Search (the perfect binary tree)



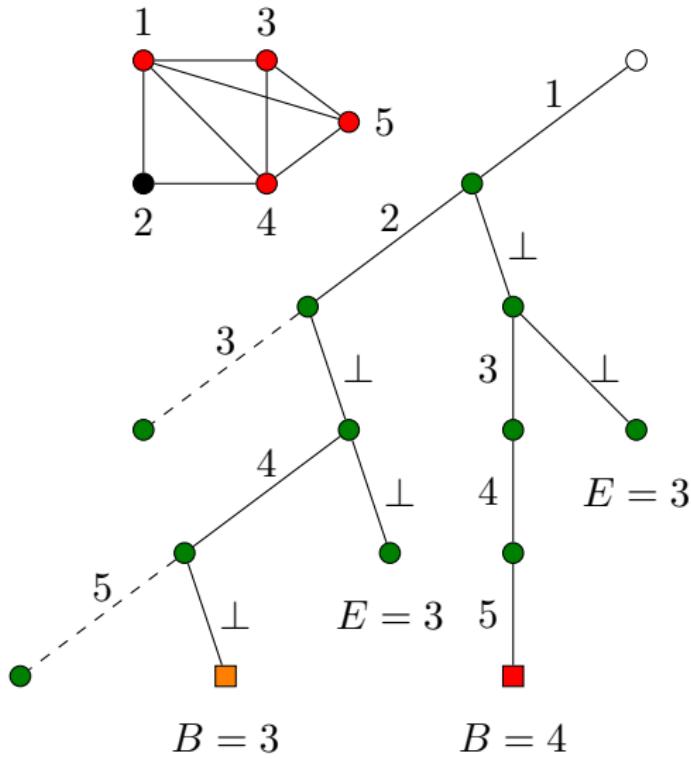
initial value $B = 0$

$\text{MC} = \{1, 2, 4\}, B = 3$

$E = 3 \leq B$, backtrack

$\text{MC} = \{1, 3, 4, 5\}, B = 4$

Demo of Search (the perfect binary tree)



initial value $B = 0$

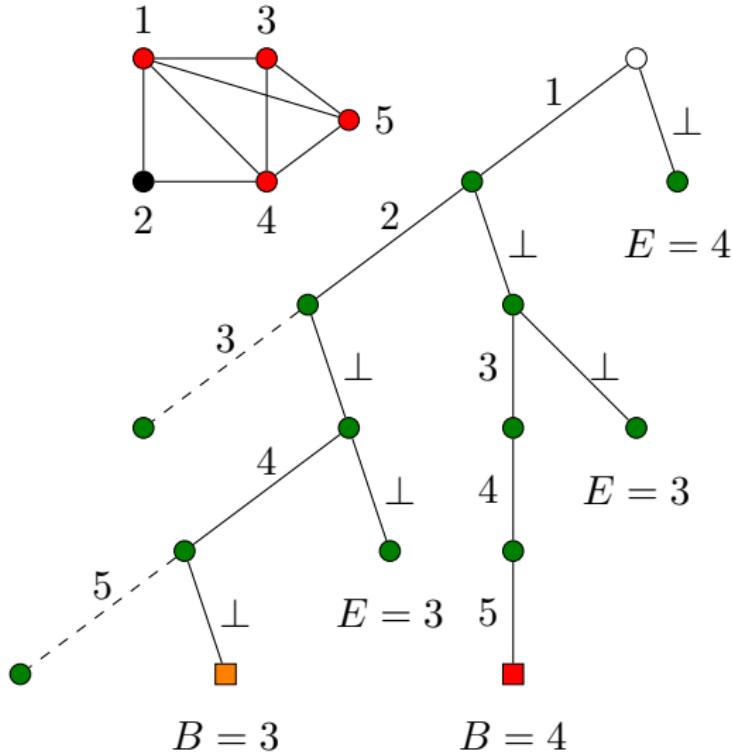
$MC = \{1, 2, 4\}, B = 3$

$E = 3 \leq B$, backtrack

$MC = \{1, 3, 4, 5\}, B = 4$

$E = 3 < B$, backtrack

Demo of Search (the perfect binary tree)



initial value $B = 0$

$MC = \{1, 2, 4\}, B = 3$

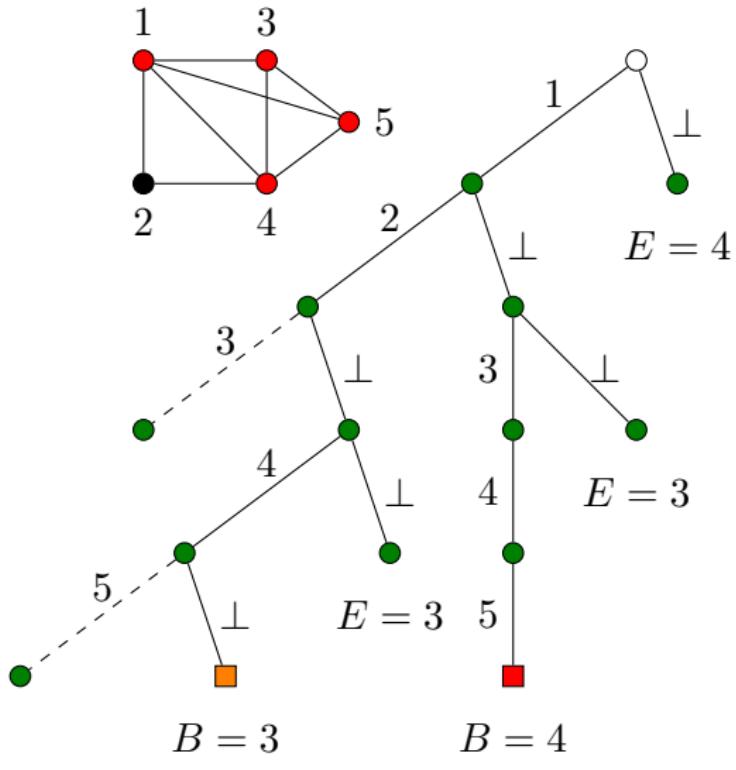
$E = 3 \leq B$, backtrack

$MC = \{1, 3, 4, 5\}, B = 4$

$E = 3 < B$, backtrack

$E = 4 \leq B$, backtrack

Demo of Search (the perfect binary tree)



initial value $B = 0$

$\text{MC} = \{1, 2, 4\}, B = 3$

$E = 3 \leq B$, backtrack

$\text{MC} = \{1, 3, 4, 5\}, B = 4$

$E = 3 < B$, backtrack

$E = 4 \leq B$, backtrack

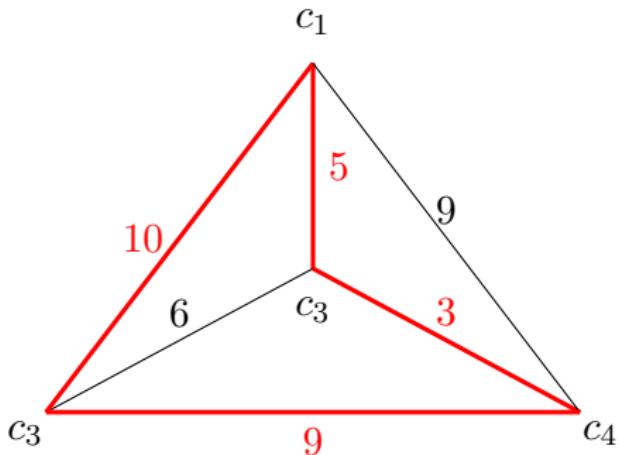
$\text{MC} = \{1, 3, 4, 5\}$

Outline

- 1 Introduction to Branch and Bound
- 2 Knapsack Problem
- 3 Maximum Clique Problem (MCP)
- 4 Traveling Salesman Problem

Traveling Salesman Problem (TSP)

Problem. Given n cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?



Modeling

Input. A finite set of cities $C = \{c_1, c_2, \dots, c_n\}$, distance $e(c_i, c_j) = e(c_j, c_i) \in \mathbb{Z}^+$, $1 \leq i < j \leq n$.

Solution. A permutation of $(1, 2, \dots, n)$ — (i_1, i_2, \dots, i_n) such that:

$$\min \left\{ \sum_{i=1}^n e(c_{k_i \bmod n}, c_{k_{i+1} \bmod n}) \right\}$$

State space. Permutation tree, node (i_1, i_2, \dots, i_k) represents route up to k steps

Constraint. Let $S = \{i_1, i_2, \dots, i_k\}$, then $i_{k+1} \in V - S$, cause every node can be visited once and only once.

Bound Value and Estimate Function

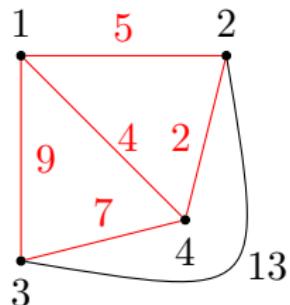
Bound Value: the length of current shortest route

Estimate Function: let the length of shortest edge connected to c_i is ℓ_i , d_j is the j -th length in the current route

$$E([i_1, \dots, i_k]) = \sum_{j=1}^{k-1} d_j + \ell_{i_k} + \sum_{i_j \notin S} \ell_{i_j}$$

- the first part: length of traveled route
- the second part: lower bound of rest route

Example of Bound Function

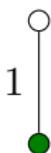


$$E([i_1, \dots, i_k]) = \sum_{j=1}^{k-1} d_j + \ell_{i_k} + \sum_{i_j \in V - S} \ell_{i_j}$$

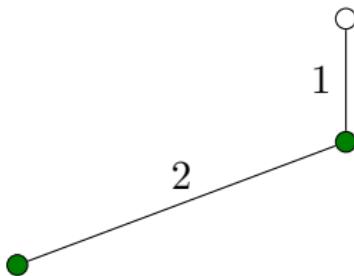
Partial route: $(1, 3, 2)$, $E([1, 3, 2]) = (9 + 13) + 2 + 2 = 26$,
 $S = \{1, 3, 2\}$, $V - S = \{4\}$

- $9 + 13$: length of traveled route
- 2: length of shortest edge connected to node 2
- 2: length of shortest edge connected to node 4

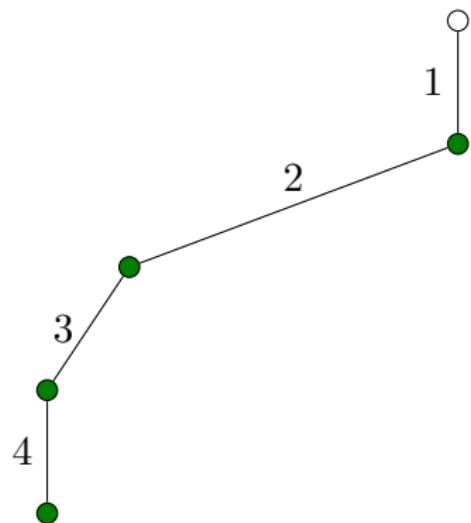
Demo of Search



Demo of Search

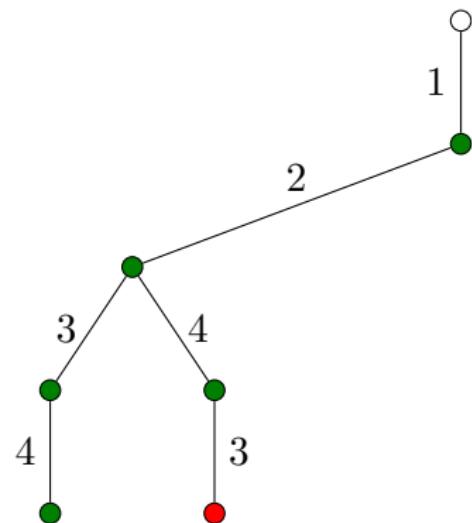


Demo of Search



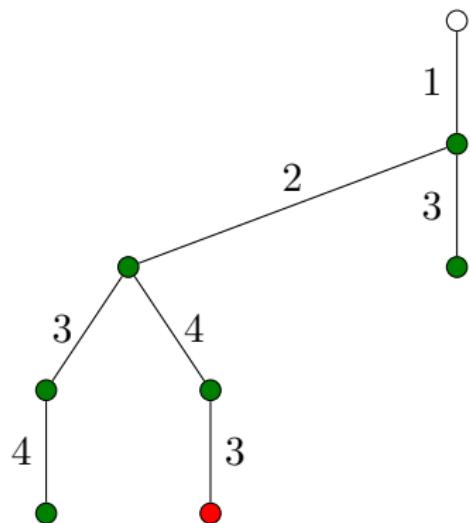
$$B = 29$$

Demo of Search



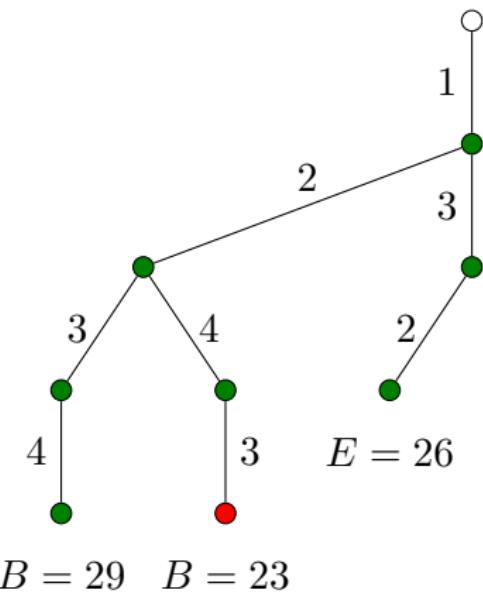
$$B = 29 \quad B = 23$$

Demo of Search

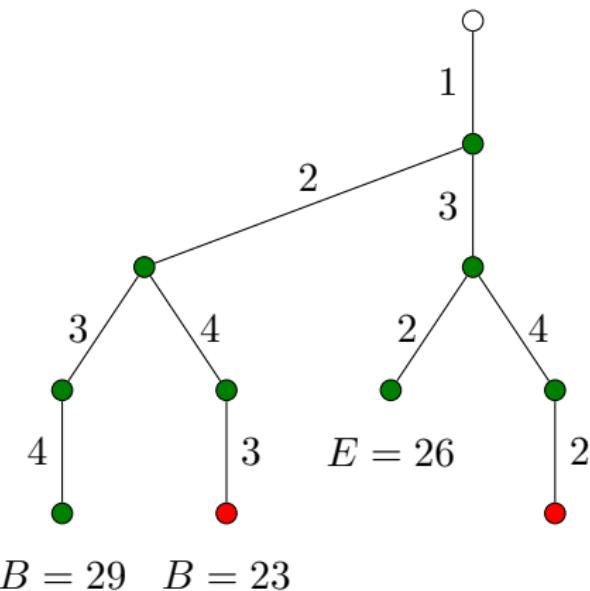


$$B = 29 \quad B = 23$$

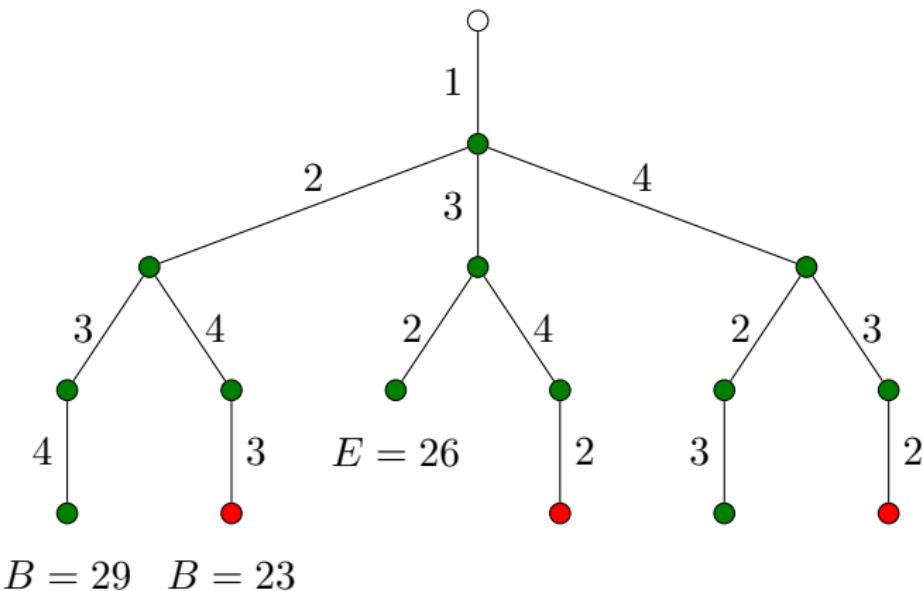
Demo of Search



Demo of Search



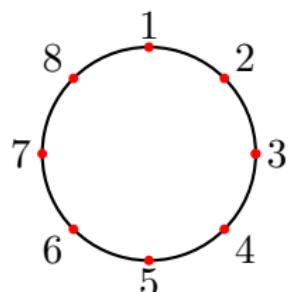
Demo of Search



Complexity Analysis (1/2)

leaf nodes: $(n - 1)!$

- each leaf node corresponds to a route
- each route (actually a circle) has n cities \leadsto equivalence under shift \leadsto at most $(n - 1)!$ different routes

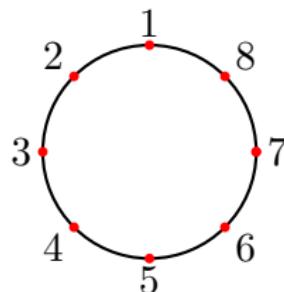
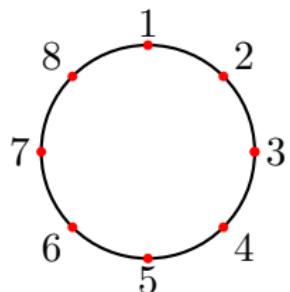


Complexity Analysis (1/2)

leaf nodes: $(n - 1)!$

- each leaf node corresponds to a route
- each route (actually a circle) has n cities \leadsto equivalence under shift \leadsto at most $(n - 1)!$ different routes

Further observation: solution is a cycle in undirected graph \leadsto clockwise and counter-clockwise are symmetric \leadsto at most $(n - 1)!/2$ different routes (two equivalences do not overlap)



Complexity Analysis (2/2)

Complexity of $E(\cdot)$ is $O(1) \rightsquigarrow$ traveling each route requires $O(n)$

$$E(i_1, \dots, i_k) = \sum_{j=1}^{k-1} d_j + \ell_{i_k} + \sum_{i_j \in V-S} \ell_{i_j}$$

Complexity Analysis (2/2)

Complexity of $E(\cdot)$ is $O(1) \rightsquigarrow$ traveling each route requires $O(n)$

$$E(i_1, \dots, i_k) = \sum_{j=1}^{k-1} d_j + \ell_{i_k} + \sum_{i_j \in V-S} \ell_{i_j}$$

- update when move to its child node i_{k+1} (add $d_k - \ell_{i_k}$),
where $d_k = e(i_k, i_{k+1})$

$$\begin{aligned} E(i_1, \dots, i_k, i_{k+1}) &= \sum_{j=1}^k d_j + \ell_{i_{k+1}} + \sum_{i_j \in V-(S+i_{k+1})} \ell_{i_j} \\ &= \sum_{j=1}^k d_j + \sum_{i_j \in V-S} \ell_{i_j} \end{aligned}$$

- the initial value is $E([i_1]) = \sum_{j=1}^k \ell_{i_j}$

Complexity Analysis (2/2)

Complexity of $E(\cdot)$ is $O(1) \rightsquigarrow$ traveling each route requires $O(n)$

$$E(i_1, \dots, i_k) = \sum_{j=1}^{k-1} d_j + \ell_{i_k} + \sum_{i_j \in V-S} \ell_{i_j}$$

- update when move to its child node i_{k+1} (add $d_k - \ell_{i_k}$),
where $d_k = e(i_k, i_{k+1})$

$$\begin{aligned} E(i_1, \dots, i_k, i_{k+1}) &= \sum_{j=1}^k d_j + \ell_{i_{k+1}} + \sum_{i_j \in V-(S+i_{k+1})} \ell_{i_j} \\ &= \sum_{j=1}^k d_j + \sum_{i_j \in V-S} \ell_{i_j} \end{aligned}$$

- the initial value is $E([i_1]) = \sum_{j=1}^k \ell_{i_j}$

The overall worse case complexity is $O(n!)$

Summary (1/3)

General steps for solving combinatorial optimization problem

- solution \sim vector
 - state space \sim search tree (partial vector is inner node, vector is leaf node)
 - searching order (DFS, BFS)
-

Brute-force algorithm: travel the entire tree (recall the integer solution for inequality problem)



Can we implement the brute-force algorithm smartly?

Summary (2/3)

Yes. The backtracking technique! Backtracking need **criteria**

Basic backtracking

- Derive the criteria from the default constraint: ensure that the Domino property holds



Additional optimization trick

- It is possible to explore symmetric property to reduce the size of the search tree

Example: loading problem, graph coloring problem

Summary (3/3)

Advanced backtracking

- Branch-and-bound method: in addition to default criteria, introduce **bound value** and **estimate function** to prune the search tree (utilizing obtained results) \leadsto further reduce the complexity \leadsto fine-grained criteria



用点子智慧

Example: MCP, TSP

When applying the branch-and-bound method, one need to find a trade-off between the gain and cost