ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

M.Sc. Program in Data Science
Department of Informatics

Optimization Techniques

Discrete Optimization

Introduction and Integer Programming Formulations

Instructor: G. ZOIS
georzois@aueb.gr

Discrete Optimization

Discrete Optimization

- So far, in all the problems we have seen,
  - We were given a function to optimize
  - The feasible region was an infinite set: A polygon, a polyhedron, R, R^n, etc

- In the rest of this course, we will see problems where
  - Input: an objective defined on some combinatorial structure, i.e., a graph, a set of numbers, some family of sets, etc
  - Constraints: they force the feasible region to be a finite set, e.g., variables can take values only in {0, 1}, or they may take integer values up to some bound

Discrete Optimization

Observation: In discrete optimization, we can always solve our problem by brute force

- Clearly not the recommended way!

We will overview techniques tailored for combinatorial structures, as

- Integer Programming algorithms (Branch and bound)
- LP-based approximation algorithms (algorithms with provable approximation guarantees)
- Local search approaches (simulated annealing)

Examples of Discrete Optimization Problems

Satisfiability – Constraint Satisfaction Problems

- <u>Boolean variables</u>: T(RUE) / F(ALSE) or 1 / 0
- <u>Boolean operators</u>: AND ($x \wedge y$), OR ($x \vee y$), NOT ($\neg x$)
- <u>Literal:</u> Boolean variable (x) or its negation ($\neg x$)
- <u>Boolean formula</u>: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$

<u>SAT</u> (decision problem)

I: a boolean formula ϕ

Q: Is ϕ *satisfiable* ?

(is there a value assignment to its variables making ϕ TRUE ?)

Example: $\phi(x,y) = (\neg x \vee y) \wedge (x \vee \neg y)$ is satisfiable

by the assignments x=y=T, and x=y=F

Examples of Discrete Optimization Problems

- **<u>Clause</u> =** A set of OR-ed literals, e.g. $(x \vee \neg y \vee z)$

- A formula is in <u>Conjunctive Normal Form (CNF)</u> if:
  - it is the **AND** of a set of clauses

E.g. $\phi = (w \vee x \vee y \vee z),\ (w \vee \overline{x}),\ (x \vee \overline{y}),\ (y \vee \overline{z}),\ (z \vee \overline{w}),\ (\overline{w} \vee \overline{z})$

Any formula ϕ can be written in CNF

<u>(CNF)-SAT</u>

I: a boolean formula ϕ in CNF

Q: Is ϕ *satisfiable* ?

One of the most fundamental problems in Computer Science

Examples of Discrete Optimization Problems

The optimization version of SAT problems:

MAX SAT

I: A CNF formula ϕ of m clauses

Q: find a truth assignment satisfying the maximum possible number of clauses

Variants of MAX SAT:

- k-CNF formula: A CNF formula where every clause has k literals (or at most k)
- Often SAT problems are stated with 3-CNF formulas
- **MAX k-SAT:** The same as MAX SAT but taking as input a k-CNF formula
- Weighted version: We can also have weights on the clauses (denoting importance of each constraint) and try to maximize total weight

Graphs

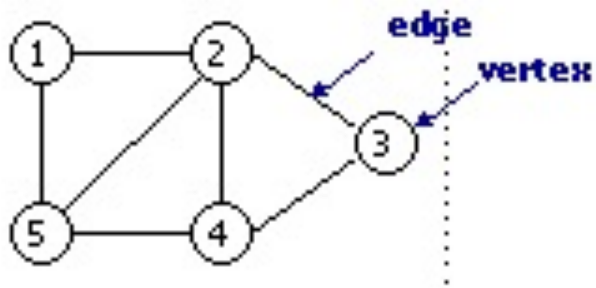
- $G = (V, E)$
- Set of nodes/vertices: $V = \{1,2,\ldots,n\}$, $|V| = n$
- Set of edges/arcs: $E \subseteq V \times V = \{(u,v) \mid u, v \in V\}$, $|E| = m$

- undirected graphs $(u,v) \equiv (v,u)$
  - $\Gamma(u) = \{v \mid (u,v) \in E\}$: neighborhood of u
  - $d(u) = |\Gamma(u)|$ = degree of u

- directed graphs $(u,v) \neq (v,u)$
  - $\Gamma^+(u) = \{v \mid (u,v) \in E\}$: out-neighborhood of u
  - $\Gamma^-(u) = \{v \mid (v,u) \in E\}$: in-neighborhood of u
  - $d^+(u) = |\Gamma^+(u)|$: out-degree of u
  - $d^-(u) = |\Gamma^-(u)|$: in-degree of u

Graph representation

- n = # vertices
- m = #edges

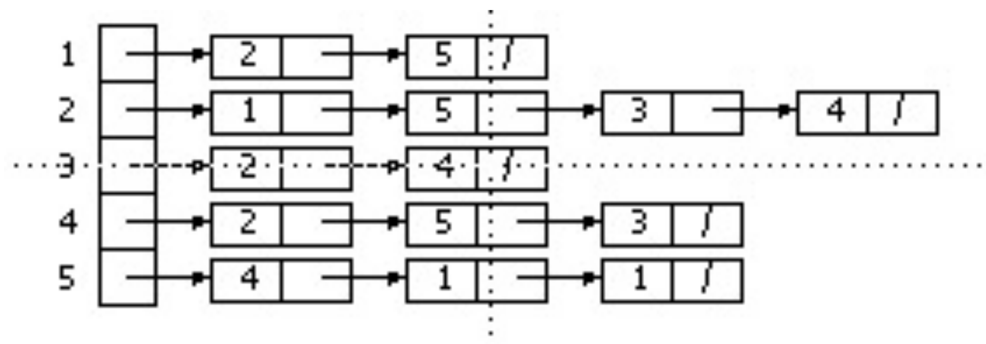


Adjacency matrix

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	0	0

space $O(n^2)$

Adjacency list



space $O(n+m)$

Dense graphs: m is $O(n^2)$

Examples of Discrete Optimization Problems

Optimization problems defined on graphs

Single-source shortest paths

I: A graph G = (V, E) with weights on its edges, and a designated vertex s
Q: The shortest paths from s to all nodes (the paths and their lengths)

Variants:

- Find shortest paths from multiple sources
- All-pairs shortest paths

Minimum Spanning Tree

I: A graph G = (V, E) with weights on its edges

Q: Find a subset of the edges $T \subseteq E$, so that the subgraph (V, T) is connected, and such that T is of minimum cost

Examples of Discrete Optimization Problems

Optimization problems defined on graphs

Traveling Salesman Problem (TSP)

I: A complete directed weighted graph G=(V,E)

Q: Find a Hamiltonian Cycle in G (a tour that goes through every node exactly once) of minimum cost

One of the most well studied problems in Computer Science, Operations Research, ...

Vertex Cover (VC):

I: A graph G = (V,E)

Q: Find a cover $C \subseteq V$ of minimum size, i.e., a set $C \subseteq V$, s.t. $\forall$ (u, v) $\in$ E, either u $\in$ C or v $\in$ C (or both)

Weighted Vertex Cover: Version with weights on the nodes

Examples of Discrete Optimization Problems

Optimization problems on sets and partitions

0-1 KNAPSACK

I: A set of objects S = {1,…,n}, each with a positive integer weight w_i, and a value v_i, i=1,…,n, along with a positive integer W

Q: find $A \subseteq S$ s.t. $\sum_{i \in A} w_i \leq W$ and $\sum_{i \in A} v_i$ is maximized

MAKESPAN

I: A set of objects S = {1,…,n}, each with a positive integer weight w_i, i = 1, …,n, and a positive integer M

Q: find a partition of S into $A_1, A_2, \ldots, A_M$, s.t. $\max_{1 \leq j \leq M} \{\sum_{i \in A_j} w_i\}$ is minimized

Useful for modeling job scheduling problems

Integer Programming

Integer Programming

What is an integer program?

- A way to model problems where some variables take integer values
- Also referred to as Integer Linear Program (ILP):
- Almost the same as Linear Programs
  - Linear objective function
  - Linear constraints

Applications:

- Comparable to applications of Linear Programming
- Operations Research
- Airline scheduling problems
- Medicine
- etc

Integer Programming Formulations

- It is not always clear how to model a problem as an integer program
- The tricky part is how to express the objective function using integer variables
- Usually: Assign a binary variable x_i to a candidate object that can be included in a solution
- Interpretation:

$$x_i = \begin{cases} 1, & \text{if item } i \text{ is in the solution} \\ 0, & \text{otherwise} \end{cases}$$

Integer Programming Formulations

Examples:

0-1 KNAPSACK

max $\Sigma_i v_i x_i$

s.t.

$\Sigma_i w_i x_i \leq W$

$x_i \in \{0,1\} \quad \forall\, i \in \{1,\ldots,n\}$

Vertex Cover

min $\Sigma_u x_u$

s.t.

$x_u + x_v \geq 1 \quad \forall\, (u, v) \in E$

$x_u \in \{0,1\} \quad \forall\, u \in V$

Integer Programming Formulations

Examples:

MAKESPAN:

- Better to think of it as a job scheduling problem
- Items correspond to jobs that should be assigned to machines
- The weight corresponds to the processing time
- How do we model that a job i is assigned to machine j?

MAKESPAN

min t

s.t.

$\Sigma_i w_i x_{ij} \leq t \quad \forall j \in \{1,\ldots,m\}$

$\Sigma_j x_{ij} = 1 \quad \forall i \in \{1,\ldots,n\}$ (every job goes to exactly one machine)

$x_{ij} \in \{0,1\} \quad \forall i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}$

Complexity of Integer Programming

- Modeling a problem as an integer program does not provide any guarantee that we can solve it

Theorem: Integer Programming is NP-complete

- In fact many problems in discrete optimization are NP-complete
- Partly due to the discrete nature
- All such problems can be reduced to SAT and vice versa

Is this the end of the world?

Coping with NP-complete problems

1. Algorithms for small instances

2. Algorithms for special cases

3. Heuristic algorithms

4. Approximation algorithms

5. Randomized algorithms

Coping with NP-complete problems

1. Small instances

If we want to run an algorithm with small instances only, then an exponential time algorithm may be satisfactory

2. Special cases

Identify families of instances where we can have an efficient algorithm, e.g., 2-SAT

3. Heuristic algorithms

Algorithms that seem to work well in practice

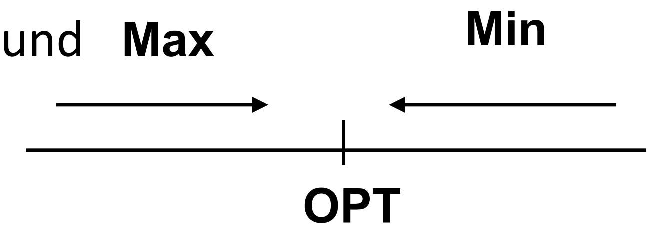
without a formal guarantee though for their performance

- Some times no guarantee that they terminate in polynomial time
- No guarantee on the approximation achieved by the solution returned

Coping with NP-complete problems

4. Approximation algorithms

Algorithms for which we can have a provable bound on the quality of the solution returned



Given an instance I of an optimization problem:

- OPT(I) = optimal solution
- C(I) = cost of solution returned by the algorithm under consideration

Definition: An algorithm A, for a minimization problem Π, achieves an approximation factor of ρ (ρ ≥ 1), if for **every** instance I of the problem, A returns a solution with:

$$C(I) \leq \rho \; OPT(I)$$

(analogous definition for maximization problems)

Coping with NP-complete problems

5. Randomized algorithms

Algorithms that use randomization (e.g. flipping coins) and take random decisions

Performance:

Such algorithms may

- produce a good solution with high probability
- produce a good cost/profit in expectation
- run in expected polynomial time

Power of randomization: for some problems, the only decent algorithms known are randomized! (e.g., primality testing)

Branch and Bound
(and related approaches)

Branch and Bound Algorithms

- A quite practical heuristic for several combinatorial problems
- Many variants over the years
- Idea: Try to avoid searching all possible solutions by keeping an estimate for the cost of the optimal solution
- Worst case: exponential, in some cases we do have to search almost all the possible solutions
- Still, average case complexity is acceptable

Backtracking

We first take a detour to a decision problem

- Consider the SAT probem
- there are 2^n possible assignments for n variables
- Going through all possible assignments yields an exponential running time: $O(2^n)$

Backtracking:

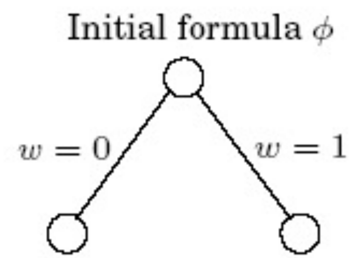
- A more intelligent exhaustive search
- Consider partial assignments
- Prune the search space
- Example:

$\phi = (w \vee x \vee y \vee z) \wedge (w \vee \neg x) \wedge (x \vee \neg y) \wedge (y \vee \neg z) \wedge (z \vee \neg w) \wedge (\neg w \vee \neg z)$

Backtracking

Start with the initial formula

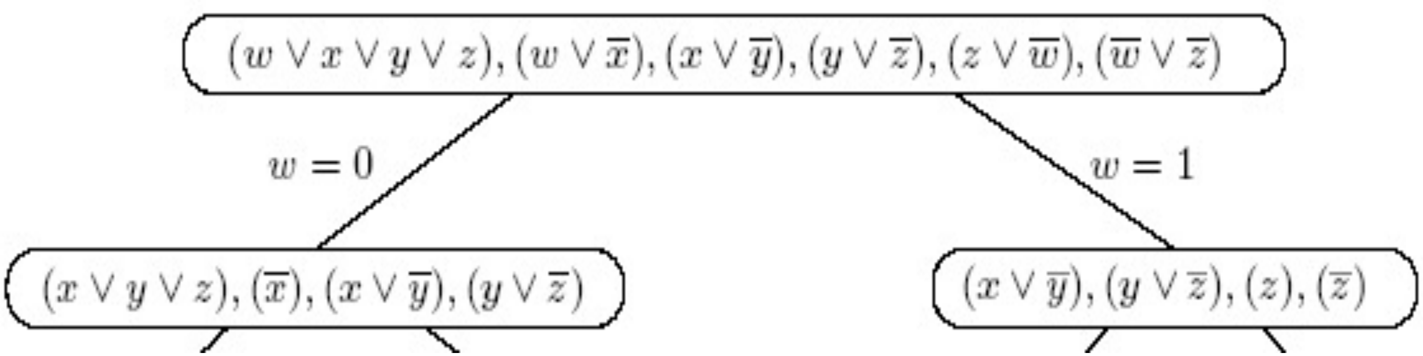
Branch on a variable, e.g. w



Plug into ϕ the values of w

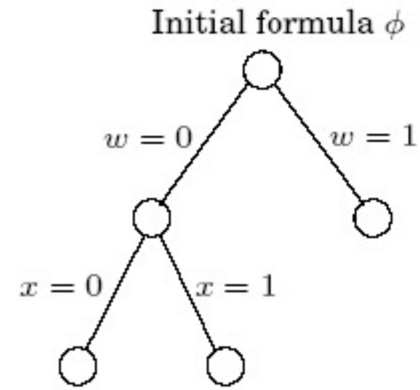
No clause is immediately violated

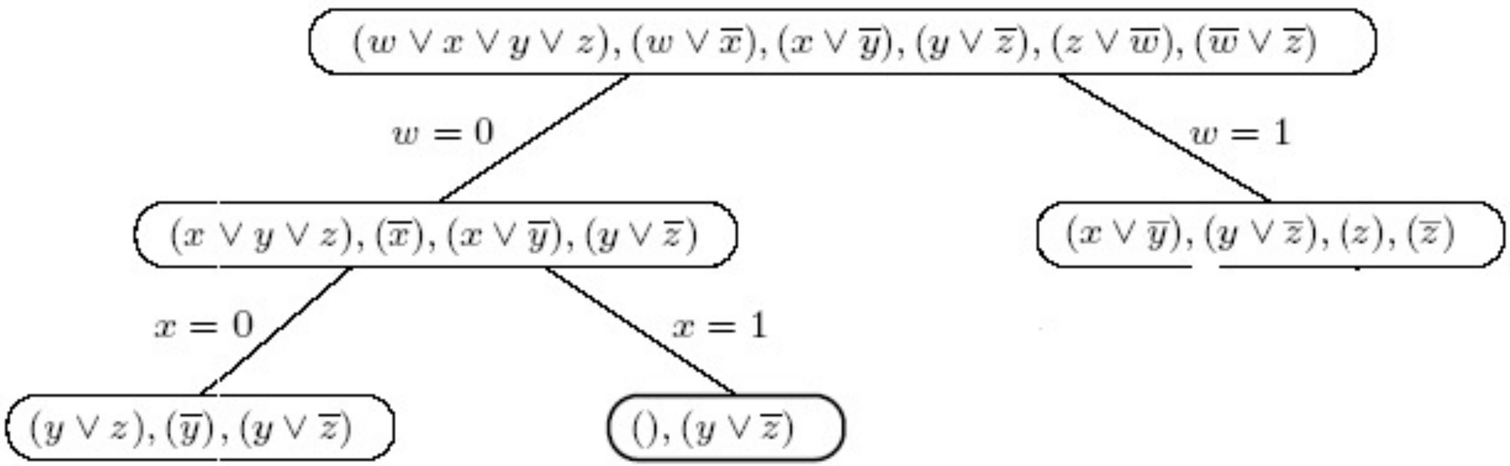
Keep active both branches



Backtracking

Expand an active node on a new variable, e.g. x



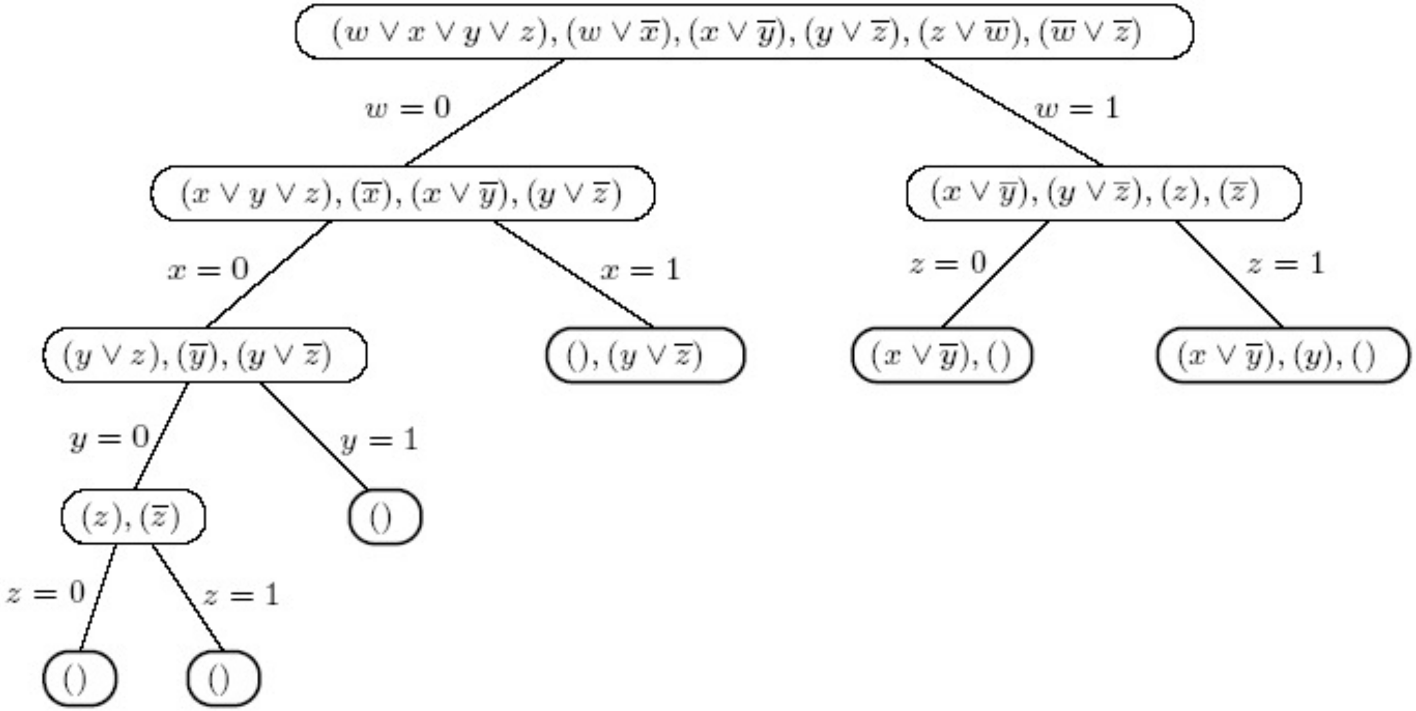


When we see ( ):

- FALSE clause; do not expand further
- the partial assignment cannot make ϕ satisfiable

Backtracking

Finally:



- The final answer to the problem is NO
- No truth assignment can satisfy ϕ
- Did not have to search all possible assignments

Branch and Bound Algorithms

From Backtracking to Branch and Bound

- This version of backtracking works well for binary problems (is the formula satisfiable or not?)
- For optimization problems, we can apply a similar approach, but taking the objective function into account
- General method, not applicable only for integer programs
- For the method to be applicable, we first need to estimate bounds on the optimal solution for various sub-instances
    - By exploiting properties of the problem at hand
- During the exploration of the solution space, we can then avoid looking at partial solutions with “high” lower or “low” upper bounds.

Branch and Bound Algorithms

Before going to integer programs, we first illustrate the general method on TSP

Traveling Salesman Problem (TSP)

I: A complete directed weighted graph G=(V,E)

Q: Find a Hamiltonian Cycle in G (a tour that goes through every node exactly once) of minimum cost

- Solution space: n!
  - Really impossible to do brute force (worse than 2^n)
- Q: How can we find a good lower bound on the cost of the optimal tour?

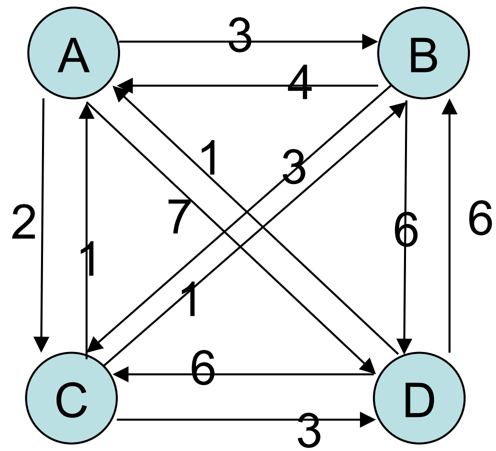
Branch-and-Bound

A lower bound on the optimal solution: $$\frac{1}{2}\sum_{i=1}^{n}(\min_{j\neq i}\{w_{i,j}\}+\min_{j\neq i}\{w_{j,i}\})$$

- the half of the sum of minimum elements of each row and each column
- For every node one edge of the tour has to come towards i and one has to leave from i

Σ_0

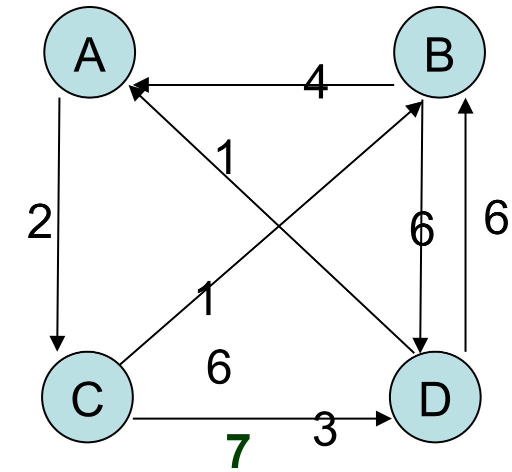
	A	B	C	D	
A	**x**	**3**	**2**	**7**	**2**
B	**4**	**x**	**3**	**6**	**3**
C	**1**	**1**	**x**	**3**	**1**
D	**1**	**6**	**6**	**x**	**1**
	1	**1**	**2**	**3**	**LB = 14/2 = 7**



Branch-and-Bound

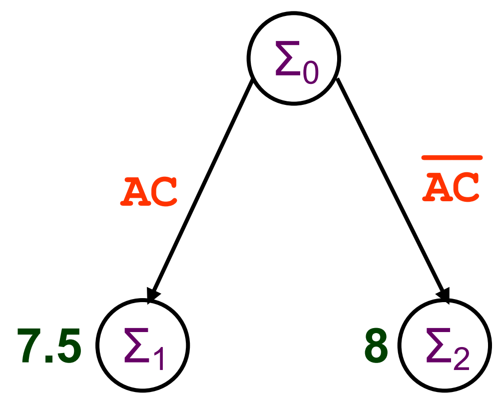
Σ_1 **Branch 1: edge AC in the tour → CA, AB, AD, BC, DC not in tour (why ?)**

	A	B	C	D	
A	x	x	2	x	2
B	4	x	x	6	4
C	x	1	x	3	1
D	1	6	x	x	1
	1	1	2	3	LB = 15/2 = 7.5



Σ_2 **Branch 2: AC not in tour**

	A	B	C	D	
A	x	3	x	7	3
B	4	x	3	6	3
C	1	1	x	3	1
D	1	6	6	x	1
	1	1	3	3	LB = 16/2 = 8



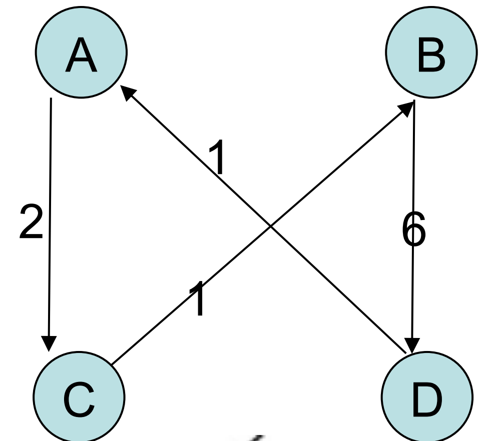
Branch-and-Bound

Σ_3

AC in tour → CA, AB, AD, BC, DC not in tour
CB in tour → CD, DB, BA not in tour

	A	B	C	D	
A	x	x	2	x	2
B	x	x	x	6	6
C	x	1	x	x	1
D	1	x	x	x	1
	1	1	2	6	LB = 20/2 = 10

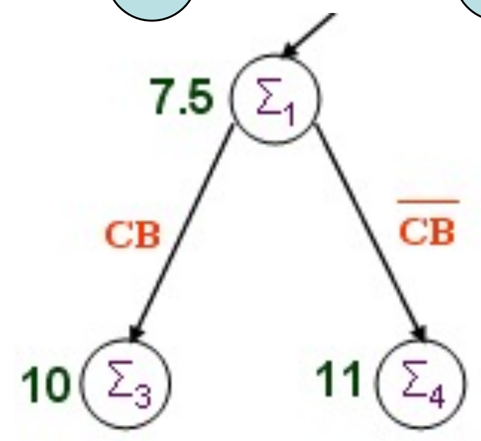
A feasible Solution



Σ_4

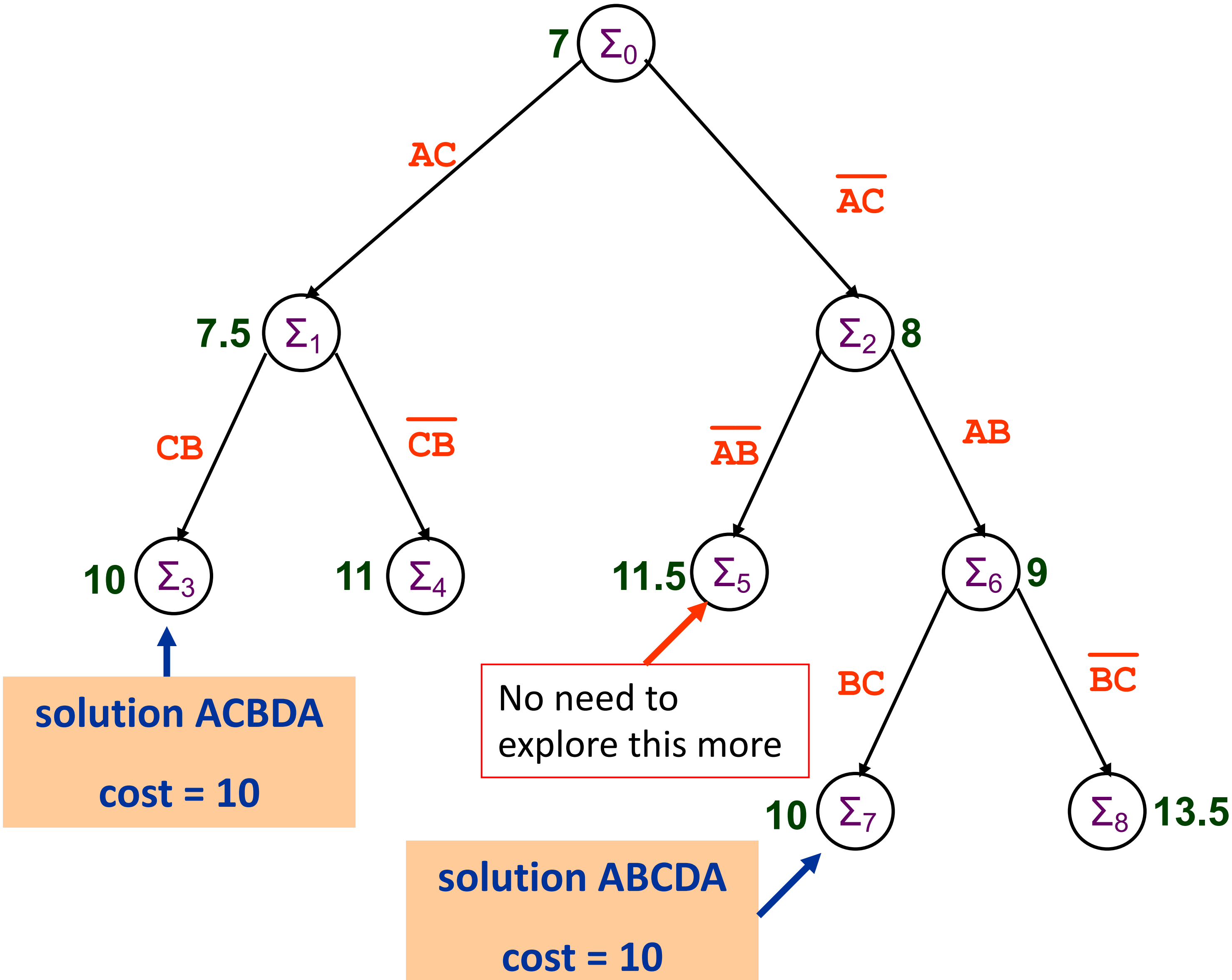
AC in tour → CA, AB, AD, BC, DC not in tour
CB not in tour

	A	B	C	D	
A	x	x	2	x	2
B	4	x	x	6	4
C	x	x	x	3	3
D	1	6	x	x	1
	1	6	2	3	LB = 22/2 = 11



and so on …

Branch-and-Bound



Branch-and-Bound

Parameters

- Maintain a set S of active states
- Initially $S = \{\Sigma_0\}$ (nothing has been expanded yet)
- In each step extract state Σ from S (Σ is the state to be expanded)
- UB is a global upper bound of the optimum solution
  - For minimization problems we initially set $UB = +\infty$
- $LB(\Sigma)$ is a lower bound on all solutions represented by state Σ (i.e. from all solutions that can arise after expanding Σ)
- Whenever we reach a terminal node with $LB(\Sigma) \leq UB$, then we can update our current UB
- During the process, we do not need to examine any further the nodes where their LB is higher than UB!

Branch-and-Bound

```
Algorithm Branch and Bound
{  S = {Σ0};
   UB = +∞
   while S ≠ ∅ do
   {    get a node Σ from S;
        //which node ? FIFO/LIFO/Best LB
        S:= S - {Σ};
        for all possible "1-step" extensions Σj of Σ do
        {      create Σj and find LB(Σj);
               if LB(Σj) ≤ UB then
                     if Σj is terminal then
                        {  UB:= LB(Σj);
                           optimum:= Σj   }
                     else add Σj to S     }      }
```

Branch and Bound for Integer Programming

- We can apply the same idea for integer programs
- Natural idea for branching: Take an integer variable and branch by setting it to either 0 or 1
- Several variants are used depending on how to choose
  - which subproblem to extract from the set of active states
  - which variable to branch on
- This has led to a wide range of very simple to very sophisticated implementations
- One of the most successful methods for solving optimally an integer program in practice
  - Very good average-case behavior

Branch and Bound for Integer Programming

Applying Branch and Bound to an integer program

- Bounding: For each subproblem we again need a bound on the optimal solution
  - How can we estimate such a bound?
  - Resort to linear programming: If we set all the remaining variables to be in [0, 1] instead of {0, 1}, the resulting problem is a LP

Definition: Consider an integer program IP where each variable $x_i \in \{0,1\}$. The LP that arises by replacing the integrality constraints with $0 \leq x_i \leq 1$ is called the LP relaxation of the IP

Theorem: Consider an integer program IP and its corresponding LP relaxation

- If IP is a maximization problem: OPT-LP ≥ OPT-IP
- If IP is a minimization problem: OPT-LP ≤ OPT-IP

Hence, we can use simplex during each iteration for the bounding step

Branch and Bound Illustrative Example

We will apply the basic variant of the technique to a maximization integer program

- A company is considering to build one new factory in Athens or Thessaloniki or in both cities
- It is also considering building a new warehouse
- Constraints:
  - The warehouse should be built in a city where a factory is also built
  - At most 1 warehouse can be built
- Every possible location for either a factory or a warehouse needs some initial capital, but also brings in some expected profitability
- Upper bound on the available capital: 10 million $

Branch and Bound Illustrative Example

Decision	Expected Profit (million $)	Capital required (million $)
Factory in Athens	9	6
Factory in Thessaloniki	5	3
Warehouse in Athens	6	5
Warehouse in Thessaloniki	4	2

Modeling the problem as an integer program

- Variables: binary variables corresponding to the decisions
  - x_1: for building a factory in Athens
  - x_2: for building a factory in Thessaloniki
  - x_3: for building a warehouse in Athens
  - x_4: for building a warehouse in Thessaloniki

Branch and Bound Illustrative Example

Constraints:

- Upper bound on the capital
  - $6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10$
- At most one warehouse
  - $x_3 + x_4 \leq 1$
- Warehouse built in a city where a factory is also built
  - $x_3 \leq x_1$
  - $x_4 \leq x_2$

Objective function

- Maximize profit
  - $9x_1 + 5x_2 + 6x_3 + 4x_4$

Branch and Bound Illustrative Example

Integer program (subproblem Σ_0):

$$\text{Max } Z = 9x_1 + 5x_2 + 6x_3 + 4x_4$$

s.t.

$$6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10$$

$$x_3 + x_4 \leq 1$$

$$x_3 \leq x_1$$

$$x_4 \leq x_2$$

$$x_i \in \{0, 1\}, i=1,2,3,4$$

Setting up branch and bound:

- Solve the corresponding LP relaxation by replacing

$$x_i \in \{0, 1\} \rightarrow 0 \leq x_i \leq 1$$

- If we get an integer solution, we are done
- Otherwise, set initial Candidate Solution (i.e., the lower bound) to $Z^* = -\infty$

Branch and Bound Illustrative Example

Integer program (subproblem Σ_0):

Max $Z = 9x_1 + 5x_2 + 6x_3 + 4x_4$

s.t.

$6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10$

$x_3 + x_4 \leq 1$

$x_3 \leq x_1$

$x_4 \leq x_2$

$x_i \in \{0, 1\}$, i=1,2,3,4

Solving the LP:

- Optimal solution = (5/6, 1, 0, 1)
- Profit = 16.5
- Hence, we have an upper bound on Σ_0, denoted as $UB(\Sigma_0)$
  - any integer solution will yield a profit of ≤ 16.5
- In fact, $UB(\Sigma_0) = 16$, since all coefficients are integers

Branch and Bound Illustrative Example

Iteration 1:

- Branching: There are many choices as to which variable to use for branching
  - Here we will just prioritize according to the index of the variable
  - First branching: $x_1 = 0$ (subproblem Σ_1) and $x_1 = 1$ (subproblem Σ_2)
  - After substitution, we have 2 new subproblems

Max $Z = 5x_2 + 6x_3 + 4x_4$

s.t.

$3x_2 + 5x_3 + 2x_4 \leq 10$

$x_3 + x_4 \leq 1$

$x_3 \leq 0$

$x_4 \leq x_2$

$x_i \in \{0, 1\}, i = 2, 3, 4$

Max $Z = 9 + 5x_2 + 6x_3 + 4x_4$

s.t.

$3x_2 + 5x_3 + 2x_4 \leq 4$

$x_3 + x_4 \leq 1$

$x_3 \leq 1$

$x_4 \leq x_2$

$x_i \in \{0, 1\}, i = 2, 3, 4$

Branch and Bound Illustrative Example

Iteration 1:

- Bounding: Need an upper bound on the optimal solution of Σ_1 and Σ_2
  - Most standard approach: Simply solve the LP relaxation of each subproblem
  - Other types of relaxations can also be used in more involved implementations

Solution to LP relaxation of Σ_1:

$(x_1, x_2, x_3, x_4) = (0, 1, 0, 1)$

With $UB(\Sigma_1) = 9$

Solution to LP relaxation of Σ_2:

$(x_1, x_2, x_3, x_4) = (1, 4/5, 0, 4/5)$

With $UB(\Sigma_2) = 16$

Branch and Bound Illustrative Example

Iteration 1:

- Final step: Check if we can dismiss any of the subproblems we have created
  - Also referred to as "fathoming"
  - We check also if we can update Z* (candidate optimal solution)

Look again at the LP relaxation of Σ_1:

- $(x_1, x_2, x_3, x_4) = (0, 1, 0, 1)$
- This is an integer solution!
- Hence we can stop this branch here, no need to explore further
- This is the optimal solution to Σ_1 itself
- Since $9 > -\infty$, update $Z^* := 9$

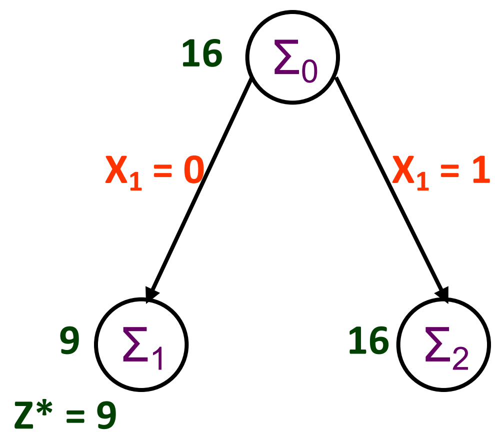
LP relaxation of Σ_2:

- $(x_1, x_2, x_3, x_4) = (1, 4/5, 0, 4/5)$
- Non-integer solution
- $16 > Z^*$
- Hence, we cannot stop here
- Need to branch further here

Branch and Bound Illustrative Example

Iteration 1:

- Summary: We can depict what we have done so far with the branching tree



Branch and Bound
Illustrative Example

When can we dismiss a node of the tree from further consideration?

1. When the solution of the LP relaxation is integer
   - As in Iteration 1
2. When the LP relaxation is infeasible
   - If the relaxation does not have a solution, there is no solution for the subproblem itself
3. When the LP relaxation results in an upper bound that is worse (i.e., less or equal) than Z*
   - In our case, if after iteration 1, we run into a subproblem Σ_i where $UB(\Sigma_i) \leq 9$, then we do not need to examine it more

Branch and Bound

Summarizing Branch and Bound for IP maximization problems

Initialization: Set $Z^* = -\infty$, check if the LP relaxation has an integer solution or if it is infeasible

In each iteration:

1. Branching: Among the remaining subproblems, pick the one created most recently
   - Break ties according to the largest upper bound
2. Bounding: Solve the LP relaxation to find an upper bound for each new subproblem
3. Checking for dismissals: For each new subproblem, check if any of the 3 criteria apply

Branch and Bound Illustrative Example

Iteration 2:

We continue from Σ_2

- Branching: We branch on whether $x_2 = 0$ or $x_2 = 1$

Subproblem Σ_3 ($x_1 = 1$, $x_2 = 0$)

Max $Z = 9 + 6x_3 + 4x_4$

s.t.

$5x_3 + 2x_4 \leq 4$

$x_3 + x_4 \leq 1$

$x_3 \leq 1$

$x_4 \leq 0$

$x_i \in \{0, 1\}$, $i = 3, 4$

Subproblem Σ_4 ($x_1 = 1$, $x_2 = 1$)

Max $Z = 14 + 6x_3 + 4x_4$

s.t.

$5x_3 + 2x_4 \leq 1$

$x_3 + x_4 \leq 1$

$x_3 \leq 1$

$x_4 \leq 1$

$x_i \in \{0, 1\}$, $i = 3, 4$

Branch and Bound Illustrative Example

Iteration 2:

- Bounding: solve the LP relaxations of Σ_3 and Σ_4

Solution to LP relaxation of Σ_3:

$(x_1, x_2, x_3, x_4) = (1, 0, 4/5, 0)$

Optimal solution: 13.8

Hence, $UB(\Sigma_3) = 13$

Solution to LP relaxation of Σ_4:

$(x_1, x_2, x_3, x_4) = (1, 1, 0, 1/2)$

Optimal solution: 16

Hence, $UB(\Sigma_4) = 16$

- Checking for dismissals (recall that $Z^* = 9$):

None of the criteria apply to Σ_3 or Σ_4

We cannot dismiss any of them at the moment

Branch and Bound Illustrative Example

Iteration 3:

- Σ_3 and Σ_4 were created during the same iteration
- We pick to continue from Σ_4, which has the largest upper bound
- Branching: We branch on whether $x_3 = 0$ or $x_3 = 1$

Subproblem Σ_5
($x_1 = 1, x_2 = 1, x_3 = 0$)

Max $Z = 14 + 4x_4$

s.t.

$2x_4 \leq 1$

$x_4 \leq 1$ (twice)

$x_4 \in \{0, 1\}$

Subproblem Σ_6
($x_1 = 1, x_2 = 1, x_3 = 1$)

Max $Z = 20 + 4x_4$

s.t.

$2x_4 \leq -4$

$x_4 \leq 0$

$x_4 \leq 1$

$x_4 \in \{0, 1\}$

Branch and Bound Illustrative Example

Iteration 3:

- Bounding: solve the LP relaxations of Σ_5 and Σ_6

Solution to LP relaxation of Σ_5:

$(x_1, x_2, x_3, x_4) = (1, 1, 0, 1/2)$

Optimal solution: 16

Hence, $UB(\Sigma_5) = 16$

LP relaxation of Σ_6:

Infeasible, first constraint cannot be satisfied

- Checking for dismissals:
  - None of the criteria apply to Σ_5
  - Σ_6 can be dismissed

Branch and Bound Illustrative Example

Iteration 4:

- We have to pick among Σ_3 and Σ_5
- We pick Σ_5 as it was created more recently
- Branching: We branch on whether $x_4 = 0$ or $x_4 = 1$
- Since this is the last variable, we can immediately read the solution

Subproblem Σ_7: $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$

- Feasible with Z = 14

Subproblem Σ_8: $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1)$

- Infeasible

Branch and Bound Illustrative Example

Iteration 4:

- Checking for dismissals:
  - First we update $Z^* = 14$ from Σ_7
  - Σ_8 is dismissed
  - We can also dismiss Σ_3, because now $UB(\Sigma_3)=13 < Z^*$

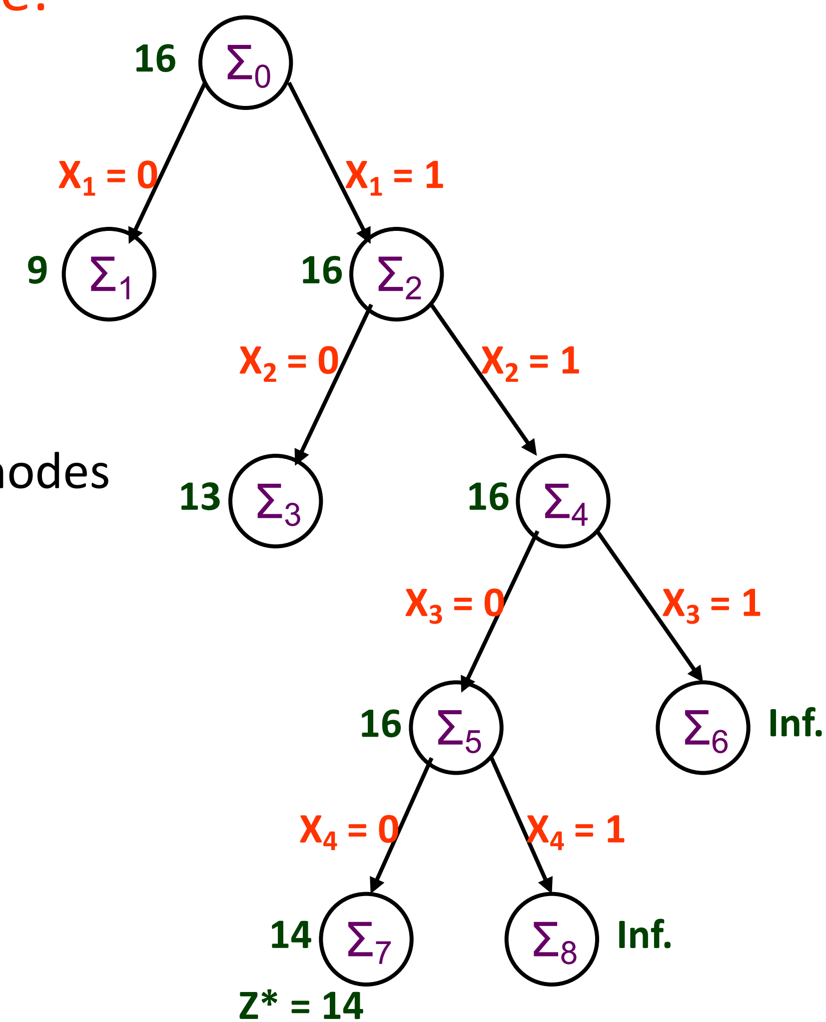
Conclusion:

Optimal solution: $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$

- Optimal profit = 14

Branch and Bound Illustrative Example

Final branching tree:



We examined only 8 nodes instead of the all 16 possible solutions

Branch and Bound

Variants and Extensions

The technique can admit numerous refinements

- Branching
  - Most popular rule is to pick the most recently created subproblem
  - Efficient because the new LP relaxation is solved by reoptimizing the previous one (small changes only)
  - Next most popular rule: Pick the subproblem with the largest upper bound
  - Branching variable: most sophisticated algorithms select the variable that is expected to produce more early dismissals
  - A popular choice: select the variable which is furthest away from being an integer in the solution of the current LP relaxation

Branch and Bound

Variants and Extensions

The technique can admit numerous refinements

- Bounding
  - The most standard way is by solving the LP relaxation
  - But any other way of "relaxing" the problem can also do
  - The Lagrangian relaxation can be used since it leads to unconstrained problems
  - Trade-off that we seek: the relaxation should be solvable relatively quickly and should also provide a relatively tight bound

Branch and Bound

Variants and Extensions

The technique can admit numerous refinements

- Finding all optimal solutions
  - The technique can be easily modified if we care to identify all optimal solutions
  - Simply need to change the way we perform dismissals and updates on Z*
- Mixed Integer Programming
  - Programs where only some variables are restricted to take integer values
  - Quite easy to adjust the technique for such cases too
  - If the integer variables are non-binary: create branches based on the possible range of the variable (e.g. $x_1 \leq 4$, and $x_1 \geq 5$)

Branch and Cut

- An even more powerful technique
- Combines branch and bound with clever preprocessing tricks
- Main extra idea: Try to reduce ("cut") the feasible region of the LP relaxations without deleting any integer solution
- Can be used to solve problems with thousands of variables
- It scales well when the constraint matrix is sparse

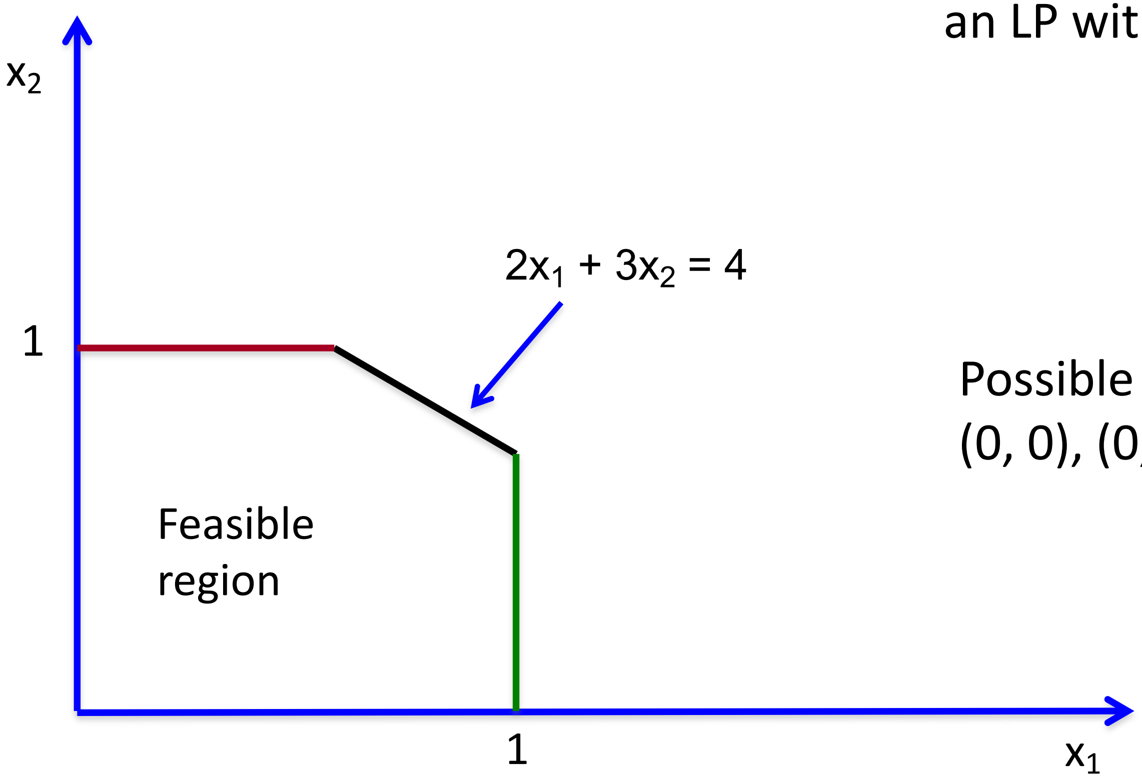
Branch and Cut

Basic steps

- Problem Preprocessing
  - Fixing variables: identify variables that can be fixed to a single value (due to the constraints)
  - Eliminate redundant constraints
  - Tighten constraints
- Generation of cutting planes
  - Reduce the feasible region of an LP relaxation without eliminating the integer solutions
- Clever branch and bound

Generating Cutting Planes

Illustration of cutting planes:



Suppose that in some iteration of branch and bound we have an LP with the constraints:

$$2x_1 + 3x_2 \leq 4$$

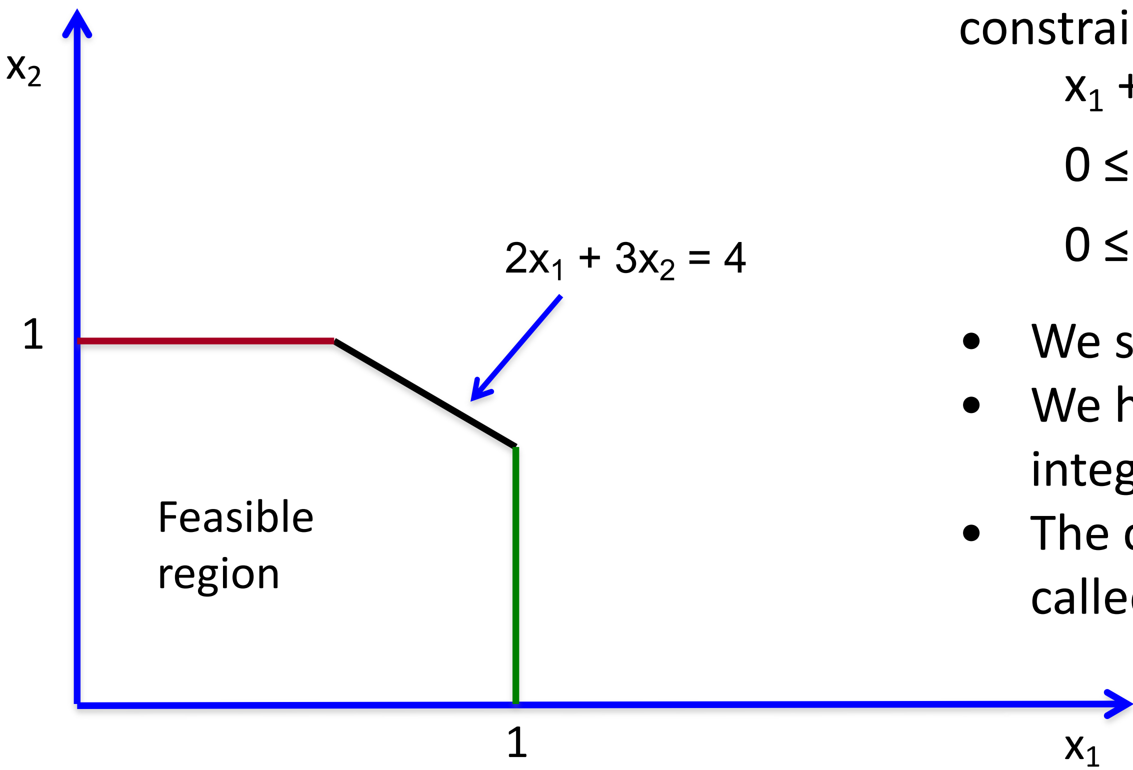
$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 \leq 1$$

Possible integer solutions:
(0, 0), (0, 1), (1, 0)

Generating Cutting Planes

Illustration of cutting planes:



Change the LP constraints to:

$$x_1 + x_2 \leq 1$$

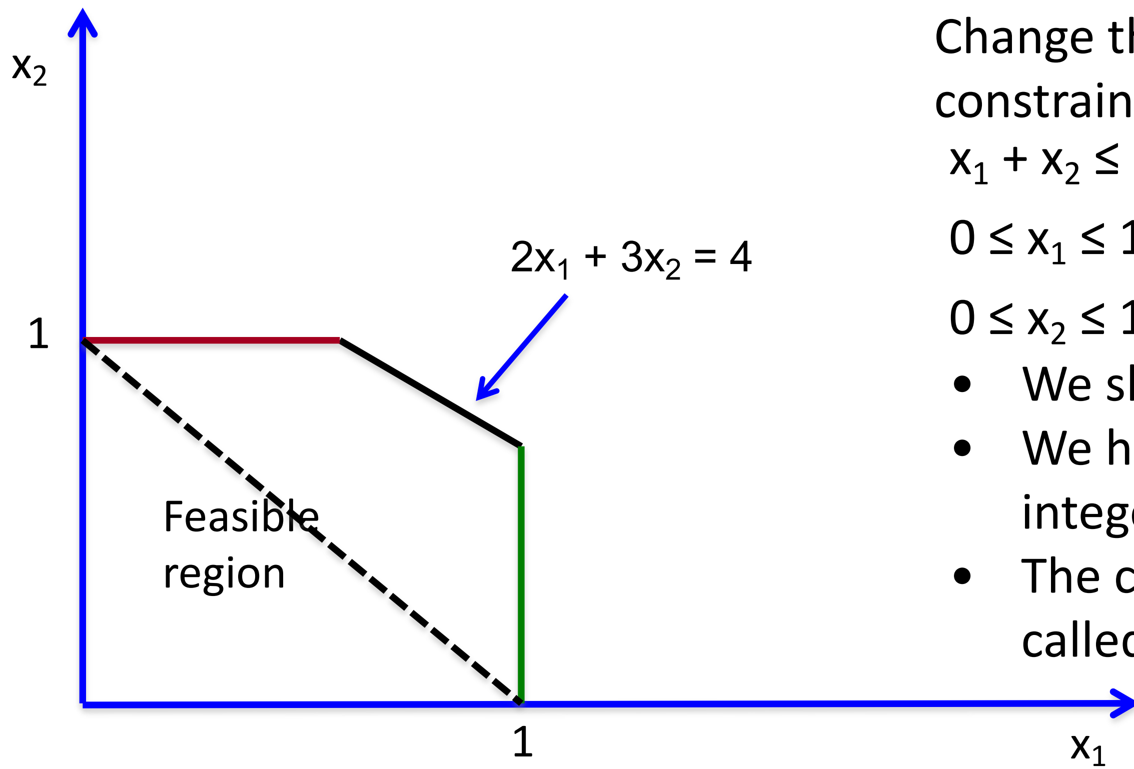
$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 \leq 1$$

- We shrank the feasible region
- We have not eliminated any integer solutions
- The constraint $x_1 + x_2 \leq 1$ is called a cutting plane

Generating Cutting Planes

Illustration of cutting planes:



Change the LP constraints to:

$x_1 + x_2 \leq 1$

$0 \leq x_1 \leq 1$

$0 \leq x_2 \leq 1$

- We shrank the feasible region
- We have not elminated any integer solutions
- The constraint $x_1 + x_2 \leq 1$ is called a cutting plane