SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

Mixed-Integer Linear Programming

CMPT 419/983
Mo Chen
SFU Computing Science
30/09/2019

Linear Programming

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \end{aligned}$$

- Convex program
  - Feasible set is a convex polytype
  - Optimal point is at vertex of feasible set (if the problem is feasible)
- Many mature solution methods
  - Simplex method
  - Interior point methods
- MATLAB: `linprog`

Mixed-Integer Linear Programming (MILP)

$$
\begin{aligned}
\text{minimize} \quad & c^\top x + d^\top y \\
\text{subject to} \quad & Ax + By \le b \\
& x, y \ge 0 \\
& x \in \mathbb{Z}
\end{aligned}
$$

- Almost same as linear program, except x must be an integer
- Very useful in many applications
  - Captures logic
- Nonconvex! → much more difficult than linear programming, or even convex programming

Integer Variables

- Binary choices
- Logical constraints
- Restricted range of values

Binary Choices

- Binary choice: $x_i \in \{0,1\}$

- Encode choice between two alternatives
  - Example: Load n items with weights w_i onto a drone with maximum weight capacity W

$$\sum_{i=1}^{n} w_i x_i \leq W$$

Logical Constraints

- Suppose $x_1, x_2 \in \{0,1\}$, where 0 represents false and 1 represent true
  - x_1 OR x_2 must be true → $x_1 + x_2 \geq 1$
  - x_1 AND x_2 must be false → $x_1 + x_2 \leq 1$

- Big-M method
  - Suppose we want $a^\top x \leq b$ OR $c^\top x \leq d$

$$
\begin{gathered}
a^\top x \leq b + My_1 \\
c^\top x \leq d + My_2 \\
y_1 + y_2 \leq 1 \\
y_1, y_2 \in \{0,1\}
\end{gathered}
$$

  - M is chosen to be very large

Restricted Range of Values

- Binary variables can be used to restrict another variable to a finite set of values

- The following are equivalent

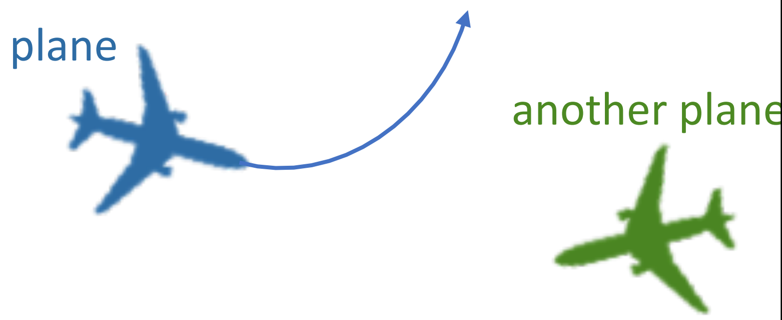
$$\boxed{x \in \{a_1, \dots, a_m\}}$$

$$x = \sum_{i=1}^{m} a_i y_i$$

$$\sum_{i=1}^{m} y_i = 1$$

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

Multi-Vehicle Collision Avoidance



- Given: algorithm for avoiding collision with another plane, despite worst-case behaviour of the other plane

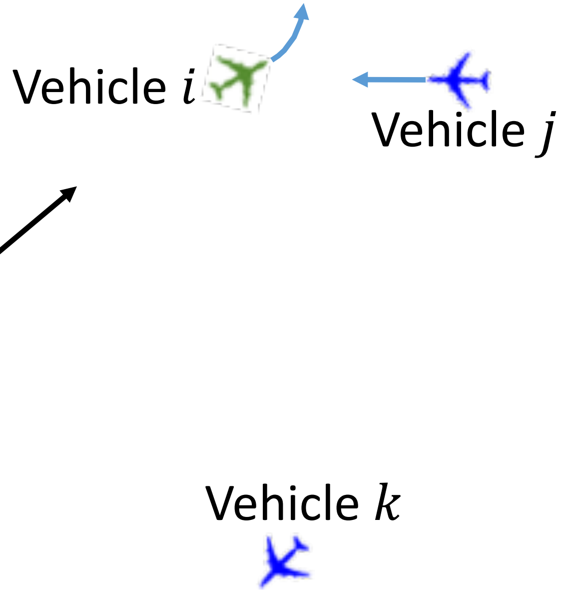
- Problem: coordinate to avoid collision when there are three or more vehicles

Chen, Shih, Tomlin (2016). Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming.

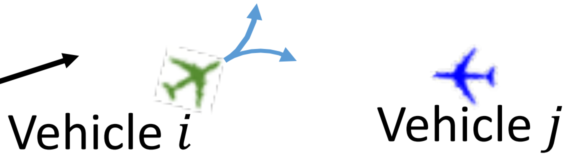
Example: Multi-Vehicle Collision Avoidance

- Only pairwise collision avoidance is tractable
  - Higher level logic is needed to guarantee collision avoidance

- Constraints
  - Vehicle j is free to avoid another vehicle
  - Vehicle i must only choose a single vehicle to avoid



- Objective
  - Resolve as many conflicts as possible

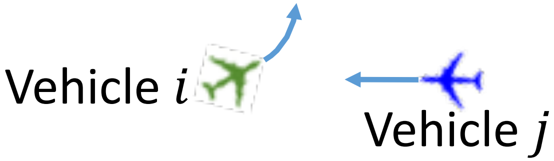


Chen, Shih, Tomlin (2016). Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming.

MVCA: Constraint Design

- **Control logic:** $\hat{u}_{ij}$, boolean variable; $\hat{u}_{ij} = 1$ if vehicle i should avoid j

$$\hat{u}_{ij} \in \{0,1\}$$

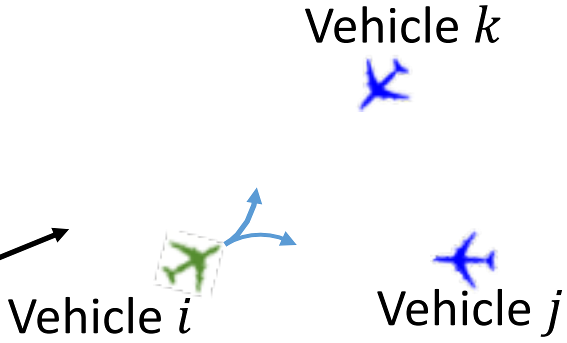


- If vehicle i avoids j, then j does not need to avoid vehicle i

$$\hat{u}_{ij} + \hat{u}_{ji} \leq 1$$

- Each vehicle i can only be guaranteed to avoid one other vehicle j

$$\sum_j \hat{u}_{ij} \leq 1$$



Chen, Shih, Tomlin (2016). Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming.

MVCA: Integer Program

Maximize $\sum_{i,j} c_{ij}\hat{u}_{ij}$

Subject to $\hat{u}_{ij} + \hat{u}_{ji} \leq 1$

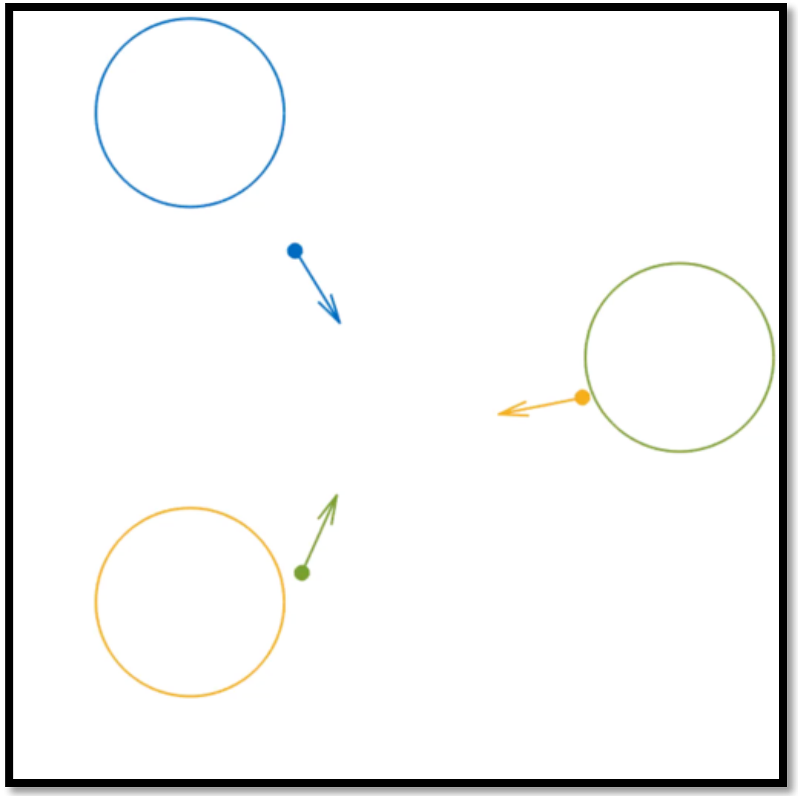
$\sum_{j} \hat{u}_{ij} \leq 1$ (No “redundant avoidance”)

$\hat{u}_{ij} \in \{0,1\}$ (Guaranteed pairwise avoidance)

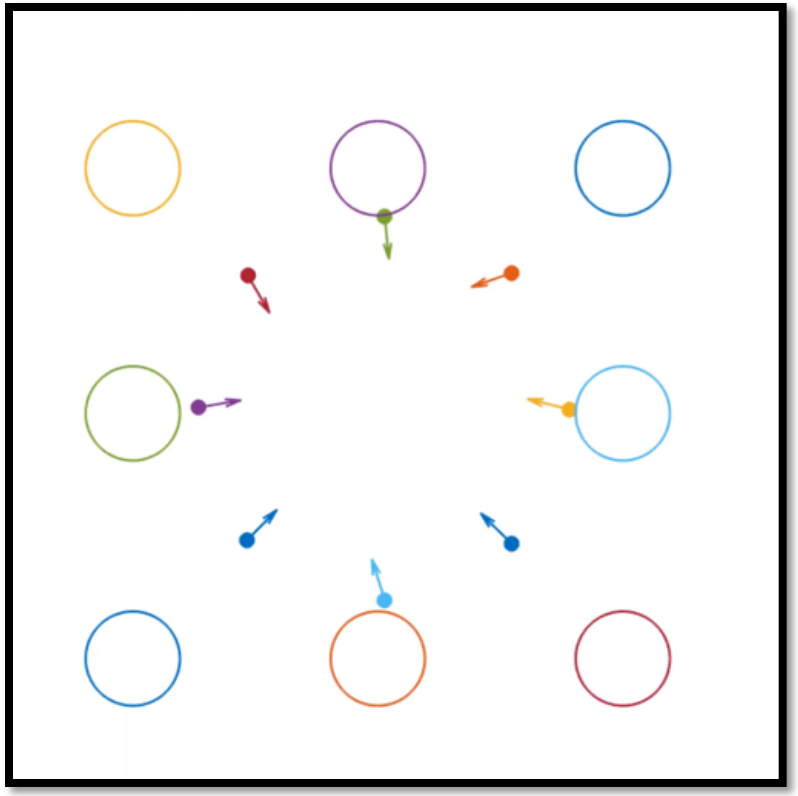
- **Reward coefficient** c_{ij}: Large c_{ij} encourages $\hat{u}_{ij}$ to be 1
- How to design c_{ij} to guarantee 3-vehicle collision avoidance?
  - Chen, Shih., Tomlin (2016). Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming.

Comparison with baseline: 3 vehicles

3 vehicles

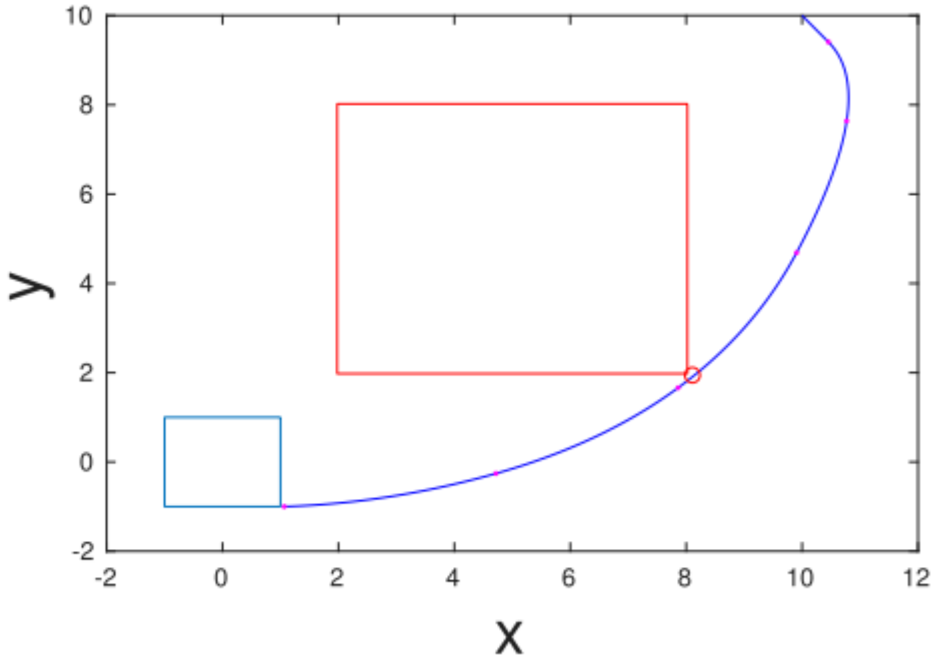


8 vehicles



Chen, Shih, Tomlin (2016). Multi-vehicle collision avoidance via Hamilton-Jacobi reachability and mixed integer programming.

Example: Obstacle Avoidance



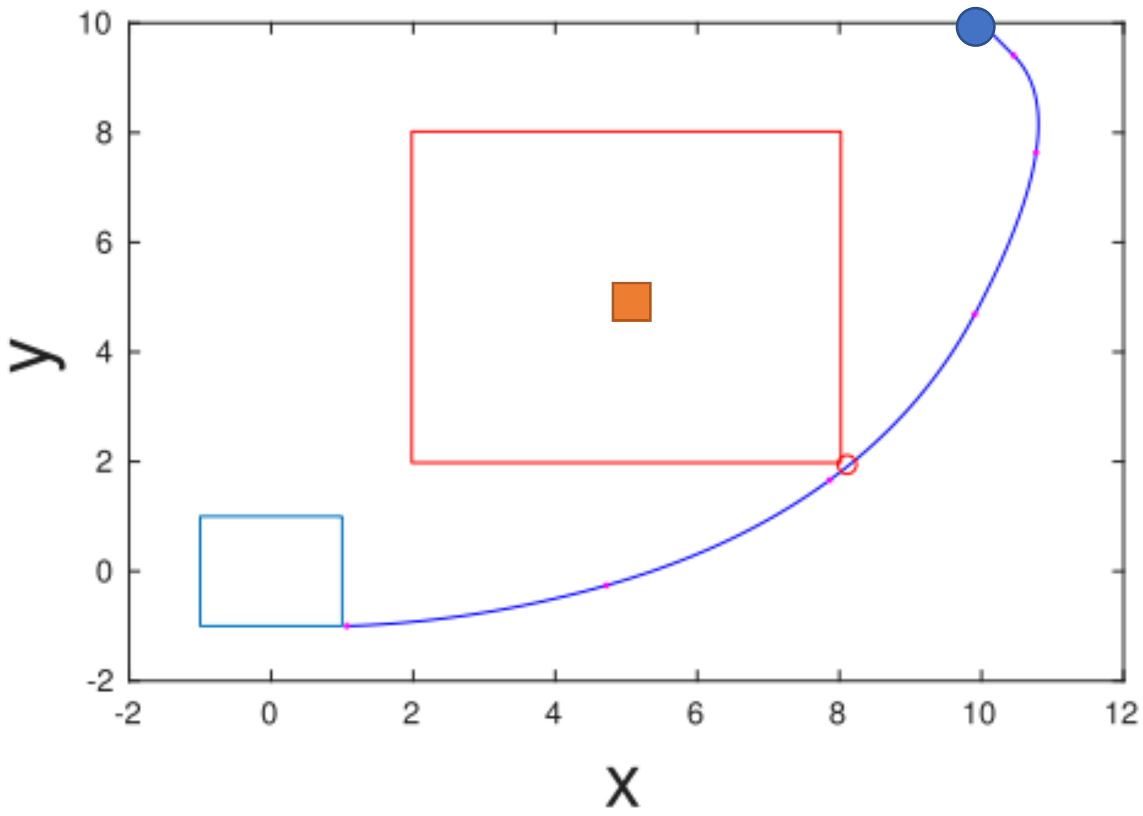
- Avoiding a box:
  - $x \leq x_l$ OR $x \geq x_u$ OR $y \leq y_l$ OR $y \geq y_u$

- Using big-M formulation:

$$\begin{aligned} x &\leq x_l + M z_1 \\ x &\geq x_u - M z_2 \\ y &\leq y_l + M z_3 \\ y &\geq y_u - M z_4 \\ z_1 + z_2 + z_3 + z_4 &\leq 3 \\ z_1, z_2, z_3, z_4 &\in \{0,1\} \end{aligned}$$

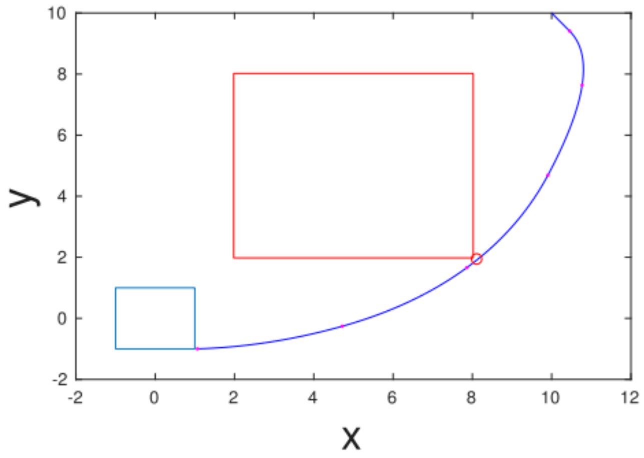
Reach-Avoid Games

- Reach a goal while avoiding an adversary

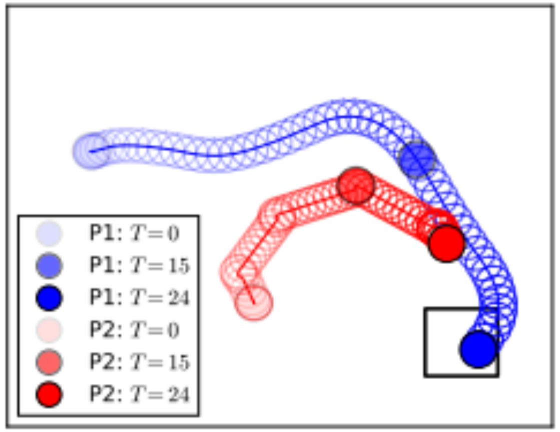


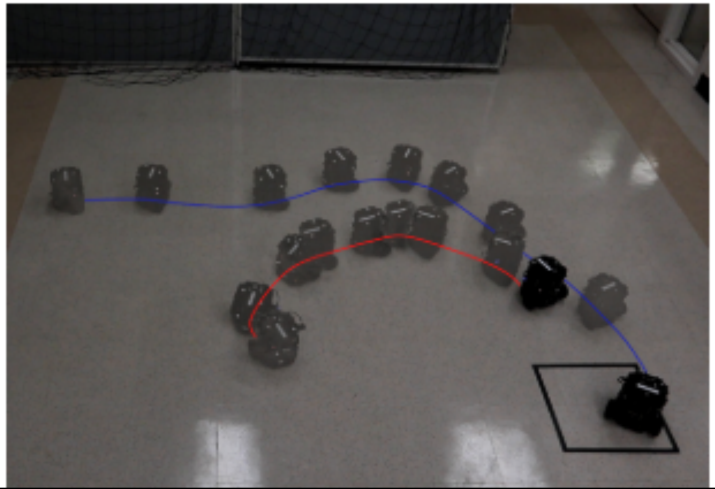
Lorenzetti, Chen, Landry, Pavone (2018). Reach-Avoid Games Via Mixed-Integer Second-Order Cone Programming.

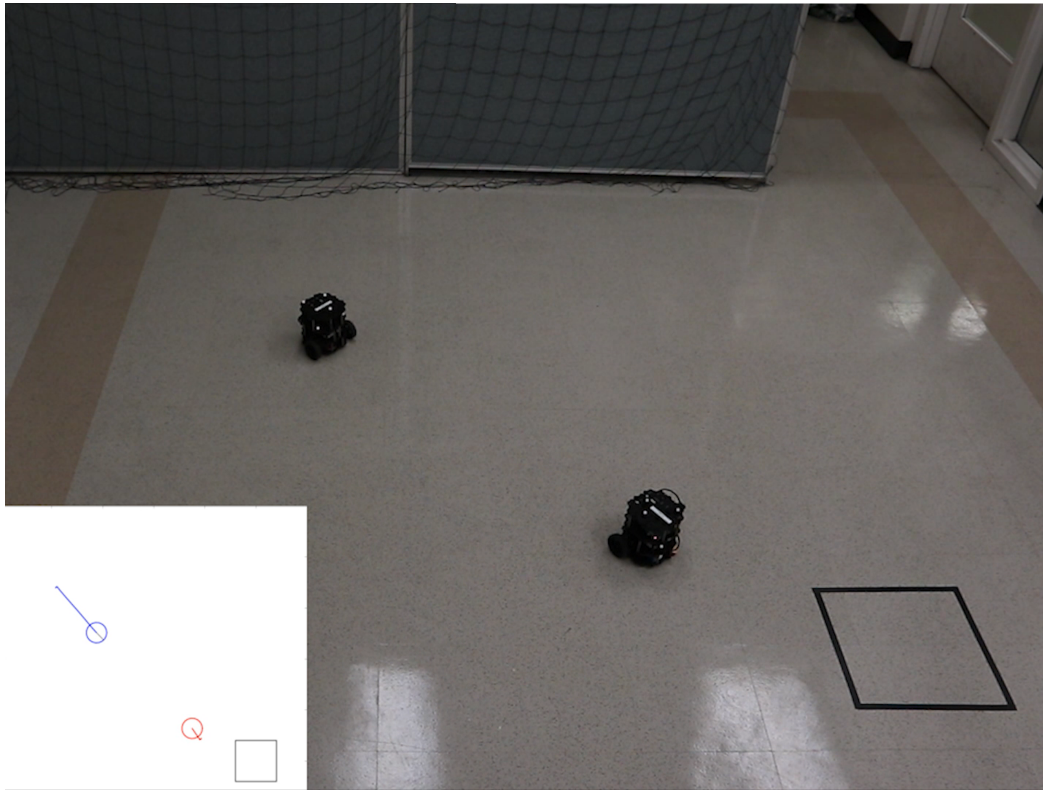
Reach-Avoid Games



- Reach a goal while avoiding an adversary







MILP Modeling

- Examples presented are not exhaustive

- Usually, there are multiple ways of modeling the same problem
    - Pick formulations that have fewer variables and constraints

- The big-M method is very popular and general, but can be hard to optimize

Solving MILPs

- Brute force approach: try all possibilities
- Branch & bound
  - Divide and conquer approach
  - Puts bound on optimal cost to eliminate possibilities quickly

Branch & Bound: Key Idea #1

- Consider the optimization

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & x \in F \end{aligned}$$

- Partition feasible set F into subsets $\{F_1, F_2, \dots, F_k\}$
  - Resulting subproblems:

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & x \in F_i \end{aligned}$$

  - Solve all subproblems, and the optimal solution to the original problem should come from one of the subproblems

Branch & Bound: Key Idea #2

- There is an easy way to obtain a **lower bound** to subproblems

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & x \in F_i \end{aligned}$$

  - Let $b(F_i)$ be the lower bound

$$b(F_i) \leq \min_{x \in F_i} c^\top x$$

- For example, solve the optimization without the integer constraint

Branch & Bound: Key Idea #3

- Maintain an upper bound U on the optimal cost of the problem.

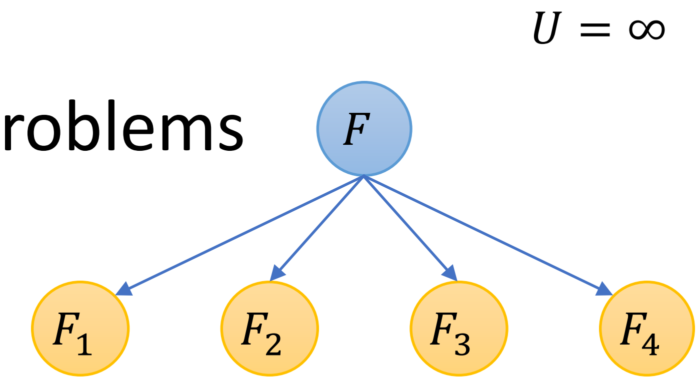
$$U \geq \min_{x \in F} c^\top x$$

  - If $b(F_i) \geq U$, then there is no need to consider the subproblem with feasible set F_i

  - U can be initialized to a very large number

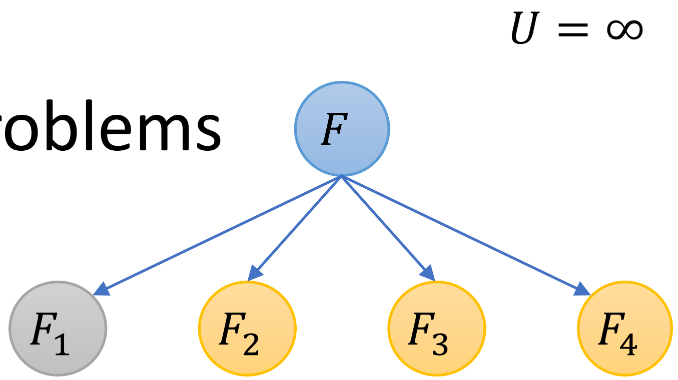
Branch & Bound Algorithm

- Initialize an upper bound U, and divide F into subproblems



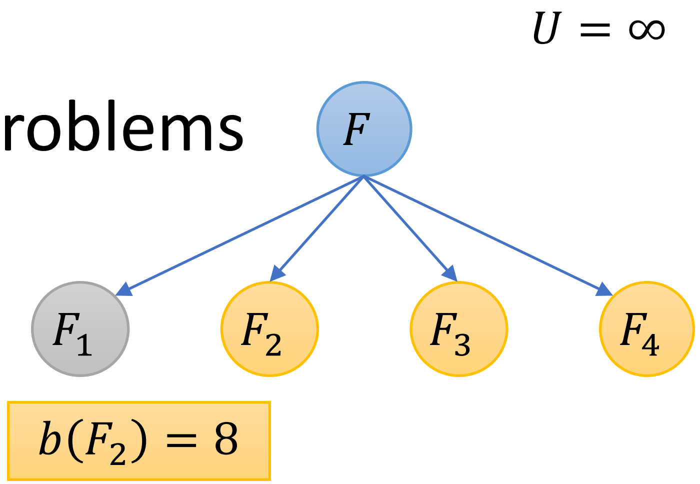
Branch & Bound Algorithm

- Initialize an upper bound U, and divide F into subproblems



1. Select an active problem
    1. If infeasible, delete it

Branch & Bound Algorithm

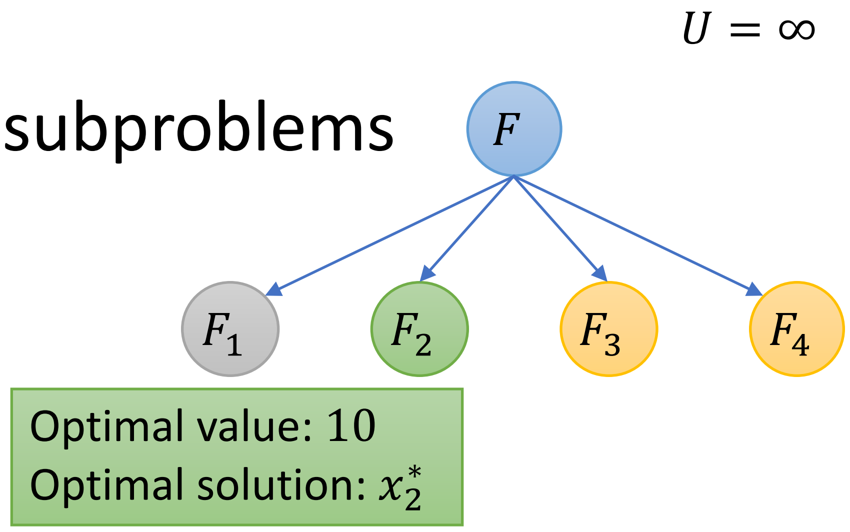


- Initialize an upper bound U, and divide F into subproblems

1. Select an active problem
   1. If infeasible, delete it
   2. Otherwise, compute lower bound $b(F_i)$

Branch & Bound Algorithm

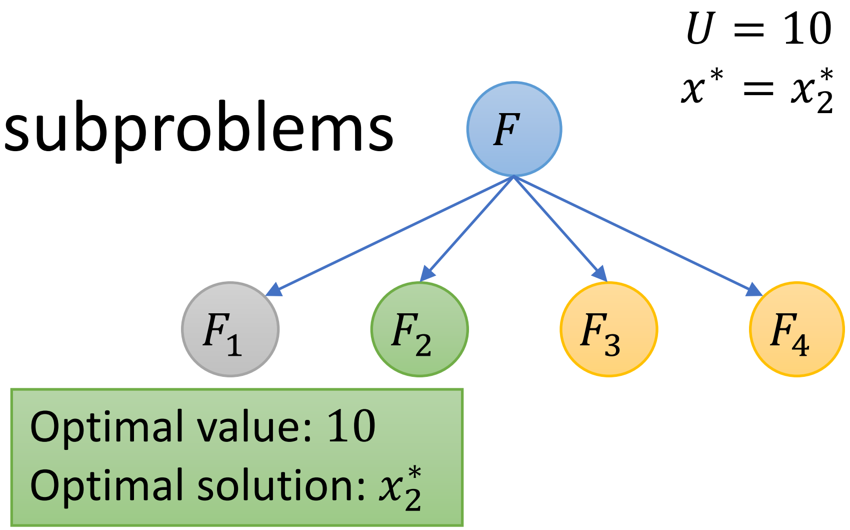
- Initialize an upper bound U, and divide F into subproblems



1. Select an active problem
   1. If infeasible, delete it
   2. Otherwise, compute lower bound $b(F_i)$
2. If $b(F_i) < U$, then two options:
   1. Solve the subproblem

Branch & Bound Algorithm

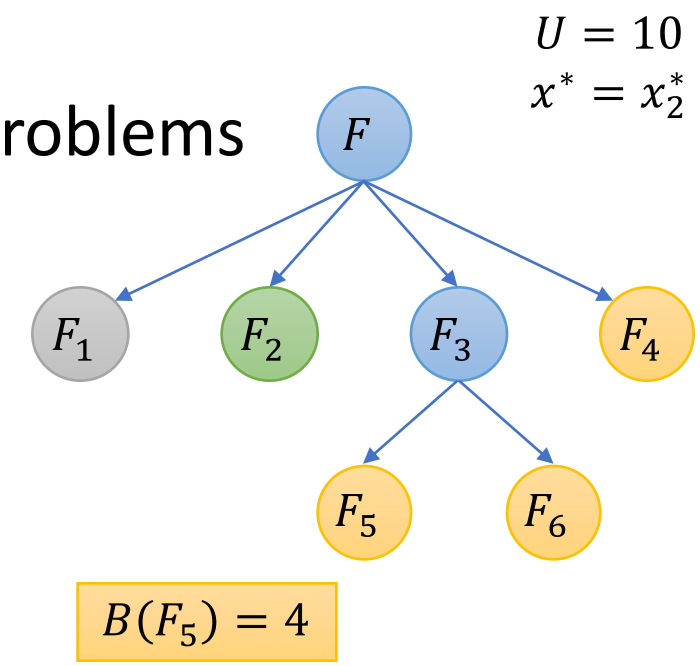
- Initialize an upper bound U, and divide F into subproblems



1. Select an active problem
   1. If infeasible, delete it
   2. Otherwise, compute lower bound $b(F_i)$
2. If $b(F_i) < U$, then two options:
   1. Solve the subproblem

Branch & Bound Algorithm

- Initialize an upper bound U, and divide F into subproblems

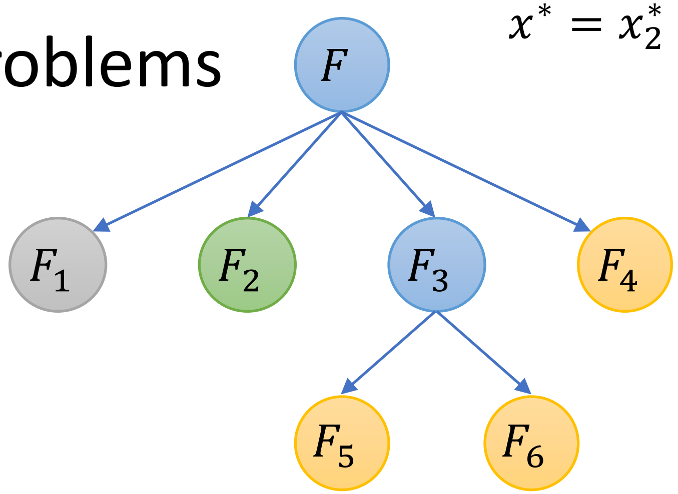


1. **Select an active problem**
    1. If infeasible, delete it
    2. **Otherwise, compute lower bound $\boldsymbol{b(F_i)}$**
2. If $b(F_i) < U$, then two options:
    1. Solve the subproblem
    2. Break the subproblem into further subproblems and add them to the list of active subproblems

Branch & Bound Algorithm

$U = 10$
$x^* = x_2^*$

- Initialize an upper bound U, and divide F into subproblems



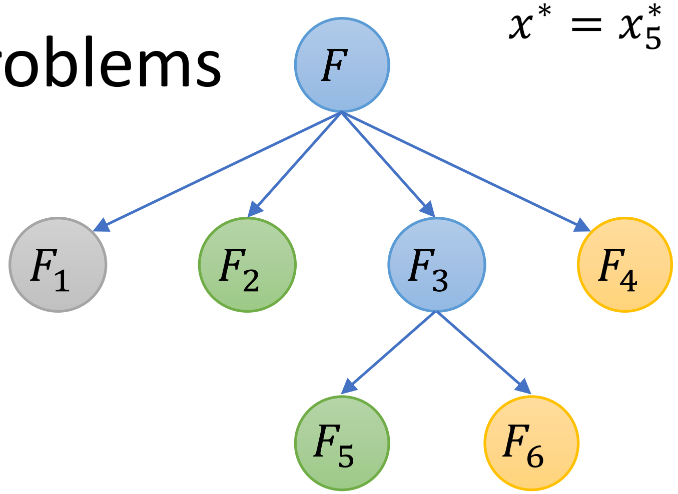
1. Select an active problem
    1. If infeasible, delete it
    2. Otherwise, compute lower bound $b(F_i)$
2. **If $b(F_i) < U$, then two options:**
    1. **Solve the subproblem**
    2. Break the subproblem into further subproblems and add them to the list of active subproblems

Optimal value: 5
Optimal solution: x_5^*

Branch & Bound Algorithm

$U = 5$
$x^* = x_5^*$

- Initialize an upper bound U, and divide F into subproblems



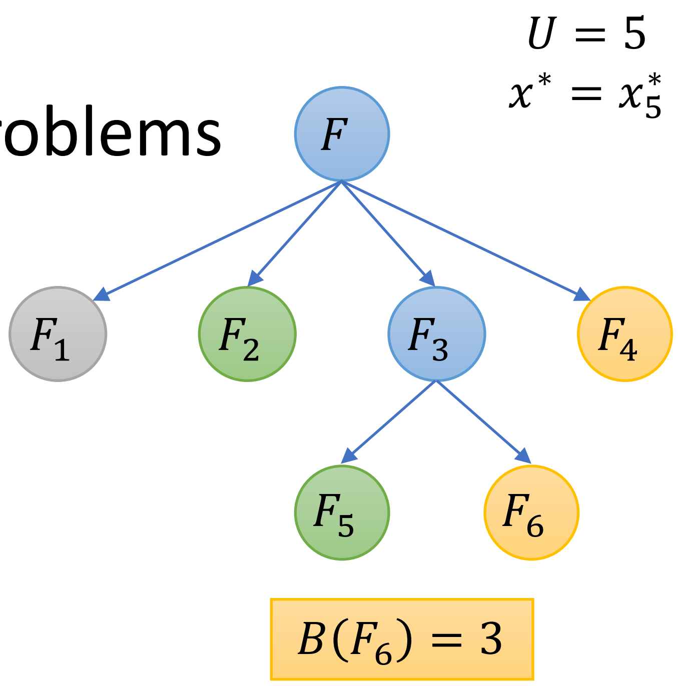
1. Select an active problem
   1. If infeasible, delete it
   2. Otherwise, compute lower bound $b(F_i)$
2. **If $b(F_i) < U$, then two options:**
   1. **Solve the subproblem**
   2. Break the subproblem into further subproblems and add them to the list of active subproblems

Optimal value: 5
Optimal solution: x_5^*

Branch & Bound Algorithm

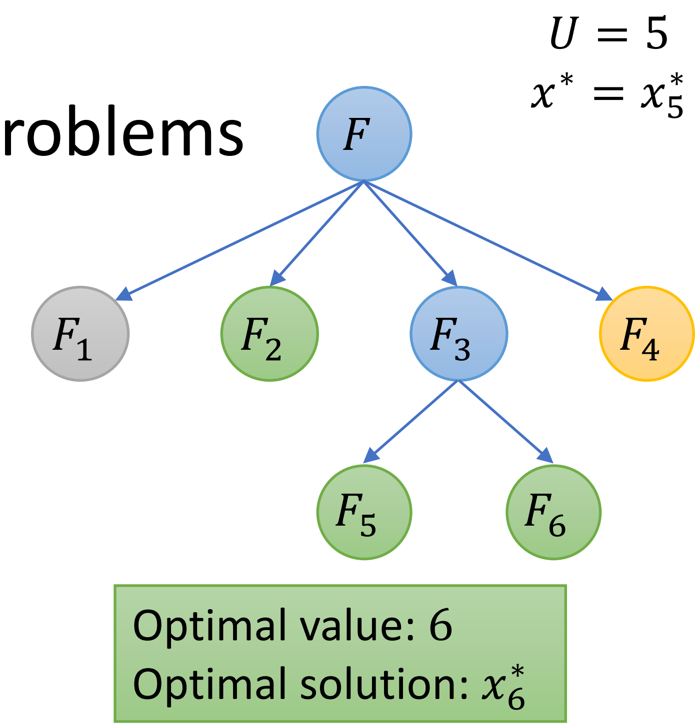
- Initialize an upper bound U, and divide F into subproblems

1. **Select an active problem**
   1. If infeasible, delete it
   2. **Otherwise, compute lower bound $\boldsymbol{b(F_i)}$**
2. If $b(F_i) < U$, then two options:
   1. Solve the subproblem
   2. Break the subproblem into further subproblems and add them to the list of active subproblems



Branch & Bound Algorithm

- Initialize an upper bound U, and divide F into subproblems

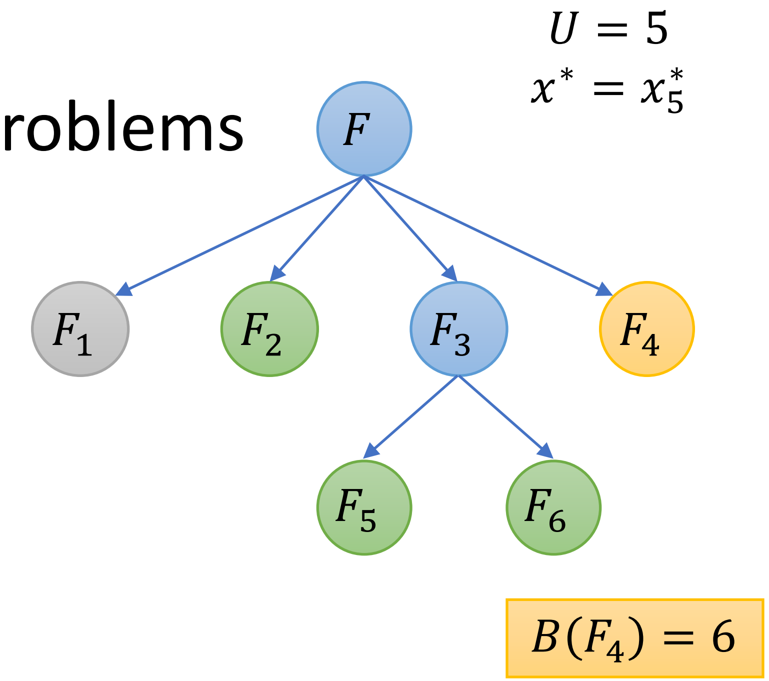


1. Select an active problem
   1. If infeasible, delete it
   2. Otherwise, compute lower bound $b(F_i)$
2. **If $\boldsymbol{b(F_i) < U}$, then two options:**
   1. Solve the subproblem
   2. **Break the subproblem into further subproblems and add them to the list of active subproblems**

Branch & Bound Algorithm

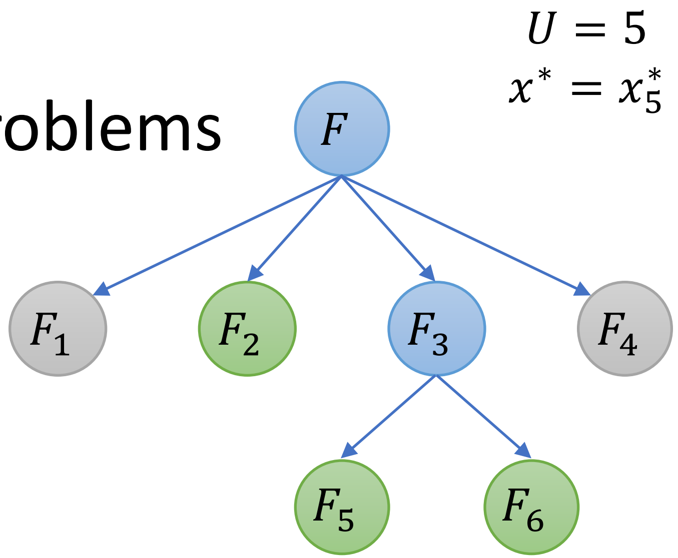
- Initialize an upper bound U, and divide F into subproblems

1. Select an active problem
    1. If infeasible, delete it
    2. Otherwise, compute lower bound $b(F_i)$
2. If $b(F_i) < U$, then two options:
    1. Solve the subproblem
    2. Break the subproblem into further subproblems and add them to the list of active subproblems
3. Otherwise, delete the subproblem



Branch & Bound Algorithm

- Initialize an upper bound U, and divide F into subproblems



1. Select an active problem
   1. If infeasible, delete it
   2. Otherwise, compute lower bound $b(F_i)$
2. If $b(F_i) < U$, then two options:
   1. Solve the subproblem
   2. Break the subproblem into further subproblems and add them to the list of active subproblems
3. Otherwise, delete the subproblem

Branch & Bound Tuning Parameters

- Choice of subproblems
  - Depth first vs. breadth first
- Different methods for obtaining lower bounds $b(F_i)$
- Different ways of breaking larger (sub)problems into smaller subproblems

Tools for Solving MILPs

- Solvers
  - CPLEX: https://www.ibm.com/analytics/cplex-optimizer
  - GLPK: https://www.gnu.org/software/glpk/
  - Gurobi: https://www.gurobi.com
  - MOSEK: https://www.mosek.com/

- Interfaces
  - Python
  - MATLAB
  - AMPL

Implementation Example

- Gurobi in Python

- Instructions if you use Anaconda: https://www.gurobi.com/documentation/8.1/quickstart_windows/installing_the_anaconda_py.html#section:Anaconda

Toy Example

- Examples/mip1.py

$$\begin{aligned} \text{maximize} \quad & x + y + 2z \\ \text{subject to} \quad & x + 2y + 3z \le 4 \\ & x + y \ge 1 \\ & x, y, z \in \{0,1\} \end{aligned}$$

```python
# Create a new model
m = Model("mip1")

# Create variables
x = m.addVar(vtype=GRB.BINARY, name="x")
y = m.addVar(vtype=GRB.BINARY, name="y")
z = m.addVar(vtype=GRB.BINARY, name="z")

# Set objective
m.setObjective(x + y + 2 * z, GRB.MAXIMIZE)

# Add constraint: x + 2 y + 3 z <= 4
m.addConstr(x + 2 * y + 3 * z <= 4, "c0")

# Add constraint: x + y >= 1
m.addConstr(x + y >= 1, "c1")

m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj: %g' % m.objVal)
```

Other Examples

- Tour of examples
  - https://www.gurobi.com/documentation/8.1/examples/example_tour.html
- Some interesting ones:
  - sudoku.py (impress your friends with this)
  - piecewise.py (piecewise linear objective)
  - portfolio.py (financial portfolio optimization)