CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Decision Analytics

Lecture 22: Duality & Mixed Integer Linear Programming

Duality

- A **dual linear program** can be defined for every linear program, which is sometimes easier to solve
- In the dual linear program the variables of the primal program become constraints and vice versa
- Also the objective function reverses direction, i.e. a primal maximisation problem becomes a minimisation problem and vice versa
- The dual of the dual program is the primal program

Duality

- The dual program is constructed as follows:

- Create a variable y_i for each constraint
- Add sign constraint "opposite" to the constraint the variable refers to, e.g. if the constraint is $a_i^T x \geq b_i$ in the primal program, the sign constraint in the dual program becomes $y_i \leq 0$
- The dual objective function is "opposite" to the primal (i.e. maximisation becomes minimisation and vice versa) and it uses the right-hand-side of the constraints as coefficients, i.e. it is $\mathrm{z} = \sum_i b_i y_i$
- The dual constraints are constructed from the transpose of the constraint matrix $A^T y$ and the coefficients of the objective function c_i connected by a comparison "opposite" to the corresponding sign constraint

Duality

- For the primal linear program

$$c^T x \rightarrow min$$

- Subject to

$$Ax \leq b$$
$$x \geq 0$$

- The corresponding dual linear program

$$b^T y \rightarrow max$$

- Subject to

$$A^T y \geq c$$
$$y \geq 0$$

Duality

- For the primal linear program

$$c^T x \to min$$

- Subject to

$$Ax \le b$$
$$x \ge 0$$

- The corresponding dual linear program

Create a variable for each constraint, use the bounds of the constraint as coefficients

$$b^T y \to max$$

- Subject to

$$A^T y \ge c$$
$$y \ge 0$$

Duality

- For the primal linear program

$$c^T x \rightarrow min$$

- Subject to

$$Ax \leq b$$
$$x \geq 0$$

- The corresponding dual linear program

$$b^T y \rightarrow max$$

- Subject to

$$A^T y \geq c$$
$$y \geq 0$$

Sign constraint "opposite" to the originating constraint

Duality

- For the primal linear program

$$c^T x \rightarrow min$$

- Subject to

$$Ax \leq b$$
$$x \geq 0$$

- The corresponding dual linear program

$$b^T y \rightarrow max$$

- Subject to

$$A^T y \geq c$$
$$y \geq 0$$

Minimisation becomes maximisation

Duality

- For the primal linear program

$$c^T x \rightarrow min$$

- Subject to

$$Ax \leq b$$
$$x \geq 0$$

- The corresponding dual linear program

$$b^T y \rightarrow max$$

- Subject to

$$A^T y \geq c$$
$$y \geq 0$$

The constraint matrix is transposed

Duality

- For the primal linear program

$$c^T x \rightarrow min$$

- Subject to

$$Ax \leq b$$
$$x \geq 0$$

- The corresponding dual linear program

$$b^T y \rightarrow max$$

- Subject to

$$A^T y \geq c$$
$$y \geq 0$$

The objective function coefficients become the bound of the constraints

Duality

- For the primal linear program

$$c^T x \rightarrow min$$

- Subject to

$$Ax \leq b$$
$$x \geq 0$$

- The corresponding dual linear program

$$b^T y \rightarrow max$$

- Subject to

$$A^T y \geq c$$
$$y \geq 0$$

The comparison operator is the opposite of the sign constraint

Example: duality

- The primal program

$$
\begin{gathered}
f = 5x_1 + 4x_2 \rightarrow max \\
x_1 \leq 4 \\
x_1 + 2x_2 \leq 10 \\
3x_1 + 2x_2 \leq 16 \\
x_1 \geq 0, x_2 \geq 0
\end{gathered}
$$

- Transforms into the dual program

$$
\begin{gathered}
w = 4y_1 + 10y_2 + 16y_3 \rightarrow min \\
y_1 + y_2 + 3y_3 \geq 5 \\
2y_2 + 2y_3 \geq 4 \\
y_1 \geq 0, y_2 \geq 0, y_3 \geq 0
\end{gathered}
$$

Strong duality

- The **strong duality theorem** states that if the value of the objective function is finite, then the optimum value of the cost functions of the primal and the dual program are identical, i.e.

$$\min_{x} c^T x = \max_{\mathrm{x}} b^T x$$

- Therefore the basic feasible solutions of the primal problem are a lower bound to the objective function, while the basic feasible solutions of the corresponding dual problem are an upper bound to the objective function
- This is true for any basic feasible solution, so even for non-optimal intermediate states bounds can be derived already
- This can be exploited in branch-and-bound algorithms that require the computation of bounds

Complementary slackness

- For optimal solutions x and y to the primal-dual pair of linear programs

$$\begin{gathered} f = c^T x \rightarrow max \\ Ax \leq b \\ x \geq 0 \end{gathered} \qquad\qquad \begin{gathered} w = b^T y \rightarrow min \\ A^T y \geq c \\ y \geq 0 \end{gathered}$$

- The following equations hold

$$\begin{gathered} y^T(b - Ax) = 0 \\ x^T(c - A^T y) = 0 \end{gathered}$$

- Because x and y are positive, this means in particular that element-by-element the inequality is tight or the corresponding dual optimal solution is zero, i.e.

$$\begin{gathered} y_i = 0 \vee (Ax)_i = b_i \\ x_i = 0 \vee (A^T y)_i = c_i \end{gathered}$$

Integer linear program

- So far we have considered linear programs on real numbers, i.e. we have allowed the variables to be $x_i \in \mathbb{R}^+$
- We will not go back to problems where the domain is again discrete, i.e. where some of the variables $x_i \in \mathbb{Z}$
- A **mixed integer linear program** is defined exactly the same then as maximising the objective function

$$f = c_0 + c^T x$$

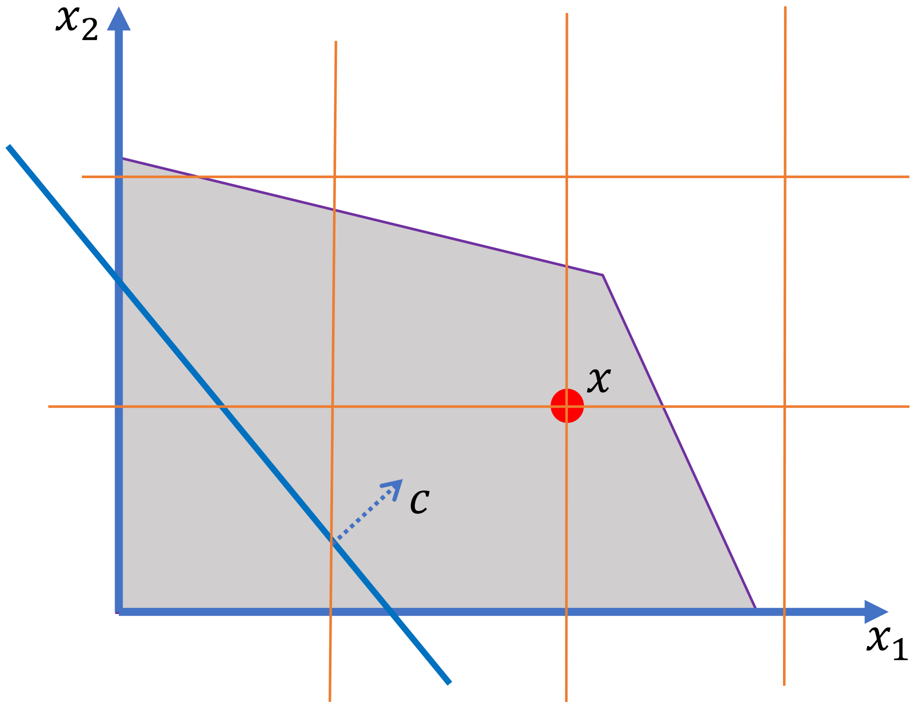
- Subject to the linear (in)equality constraints such as

$$Ax \leq b$$
$$x \geq 0$$

- With some $x_i \in \mathbb{R}$ and some $x_j \in \mathbb{Z}$

Geometric intuition

- The discretisation imposes a raster over the solution space
- The feasible solution is no longer guaranteed to be on the border of the simplex



Integer Linear Programs and SAT

- We can use ILPs to solve SAT models with linear objective function

- A Boolean variable can be encoded as $x \in \mathbb{Z}$ with the constraints

$$x \geq 0$$
$$x \leq 1$$

- Every DNF constraint

$$x_1 \vee \cdots \vee x_m \vee \neg x_{m+1} \vee \cdots \vee \neg x_n$$

- Can be encoded as

$$x_1 + \cdots + x_m + (1 - x_{m+1}) + \cdots + (1 - x_n) > 0$$

- Therefore, ILP can be considered an alternative approach to constraint programming for some discrete optimisation problems

Relaxation of an MILP

- The idea of relaxation is to drop the discretisation constraints and consider the problem as a linear program with real domain instead
- More formally, denoting the constraints as $C \subset \mathbb{R}^n$ and the subset of discrete variables as $J \subset \{1, \dots, n\}$, the MILP

$$c^T x \to max$$

subject to

$$x \in C$$
$$x_j \in \mathbb{Z}, j \in J$$

- Is relaxed to the linear program

$$c^T x \to max$$

subject to

$$x \in C$$

Relaxation of an MILP

- When we solve the relaxed linear program there are three potential outcomes:

1. The relaxed linear program is infeasible, then the MILP is infeasible as well.

2. The relaxed linear program has a solution and $\forall j \in \mathrm{J}\colon x_j \in \mathbb{Z}$, then this is the solution of the MILP as well.

3. The relaxed linear program has a solution, but $\exists j \in J\colon x_j \notin \mathbb{Z}$. This means there exists a solution to the MILP, but we still need to try to find it.

Branching

- Let's assume the integer variable x_j has the solution $\hat{x}_j \notin \mathbb{Z}$ in the relaxed IP and has therefore caused the solution not to be feasible in the MILP
- The idea is to create two new linear programs, where we add additional constraints on this variable excluding the interval between $\lfloor \hat{x}_j \rfloor$ and $\lfloor \hat{x}_j \rfloor + 1$
- This means solving two augmented linear programs with the constraints

$$\mathrm{C}' = \mathrm{C} \cup \left\{ x_j \leq \lfloor \hat{x}_j \rfloor \right\}$$

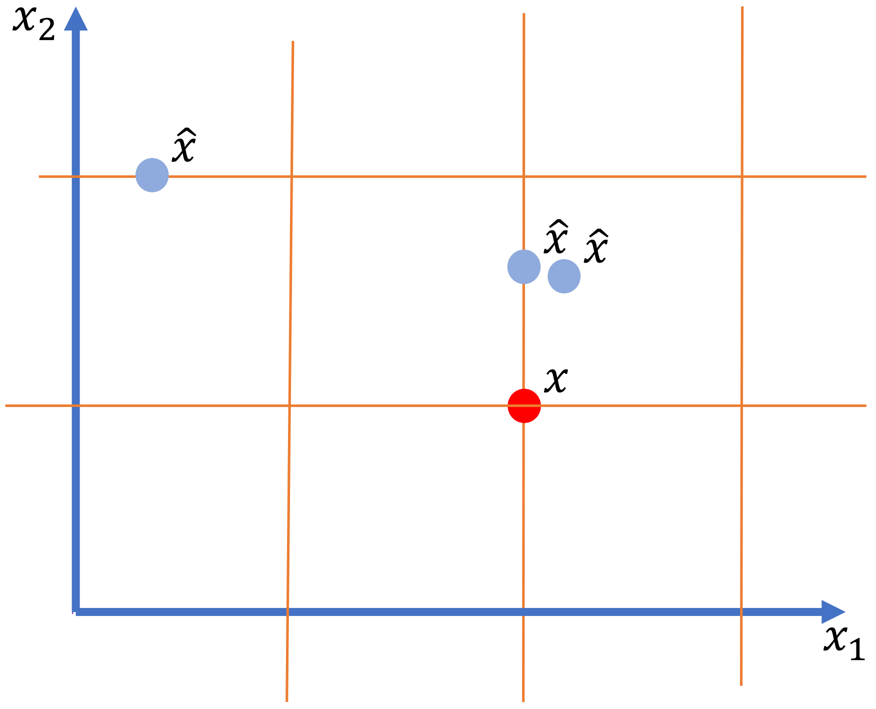
and

$$\mathrm{C}'' = \mathrm{C} \cup \left\{ x_j \geq \lfloor \hat{x}_j \rfloor + 1 \right\}$$

- This branching strategy can be recursively applied and will eventually yield a feasible solution

Geometric intuition

- The solution to the relaxed LP is infeasible on the MILP, so we add $x_1 \leq 2$
- This is still infeasible, so we add $x_2 \geq 2$
- This goes nowhere, so we decide to backtrack
- Exploring the second branch $x_2 \leq 1$ finds the optimal solution



Upper and lower bounds

- First we note, that the solution of the relaxed linear program provides an **upper bound on the objective function**, because the constraints are less stringent

- Finding a lower bound is more tricky, but there are some reasonable heuristics:
  - We could round all fractional integer variables, then fix all integer variables, and find a solution for the non-integer variables by solving the resulting linear program
  - Alternatively, we could round and fix only a subset of integer variables, then solve the resulting linear program and iterate if necessary
- Although not guaranteed, if any of these heuristics yields a feasible solution then this is a **lower bound on the objective function**

Branch and bound

- If we find a lower bound in a node of the search tree that is larger than the upper bound of the overall problem, then we can conclude that there cannot be a solution in any subtree
- We can therefore prune this sub-tree and backtrack
- Increasing the lower bounds quickly is therefore important to ensure good performance
- Every pivot of the simplex algorithm improves the lower bound, therefore we can potentially detect an infeasible branch while solving the relaxed linear program without having to iterate until the end

The Knapsack problem

Given a knapsack with a capacity to hold M kg and a set of items with weights $w_1, \dots, w_n$ and a value $v_1, \dots, v_n$, what is the maximum value we can carry without exceeding the capacity limit.

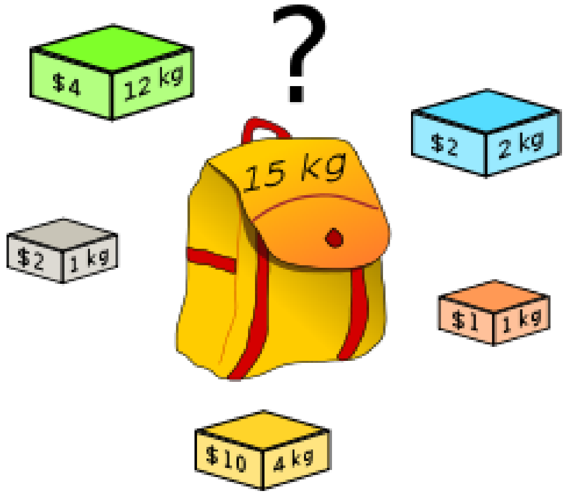
More formal we look for $x_i \in \{0,1\}$ to

- Maximise the value

$$\mathrm{V} = \sum x_i v_i$$

- Subject to the capacity constraint

$$\mathrm{W} = \sum x_i w_i \leq M$$



Knapsack using MILP

```
from ortools.linear_solver import pywraplp
solver = pywraplp.Solver('LPWrapper',
                pywraplp.Solver.CBC_MIXED_INTEGER_PROGRAMMING)

knapsack_size = 10
items = [(9,1),(2,1),(3,1)]

weight_in_knapsack = solver.Constraint(0, knapsack_size)
value_in_knapsack = solver.Objective()
for item in items:
    x = solver.BoolVar(str(item))
    weight_in_knapsack.SetCoefficient(x, items[0])
    value_in_knapsack.SetCoefficient(x, item[1])

value_in_knapsack.SetMaximization()
solver.Solve()
```

Instantiate the MILP wrapper

Create the capacity constraint

Create the objective function

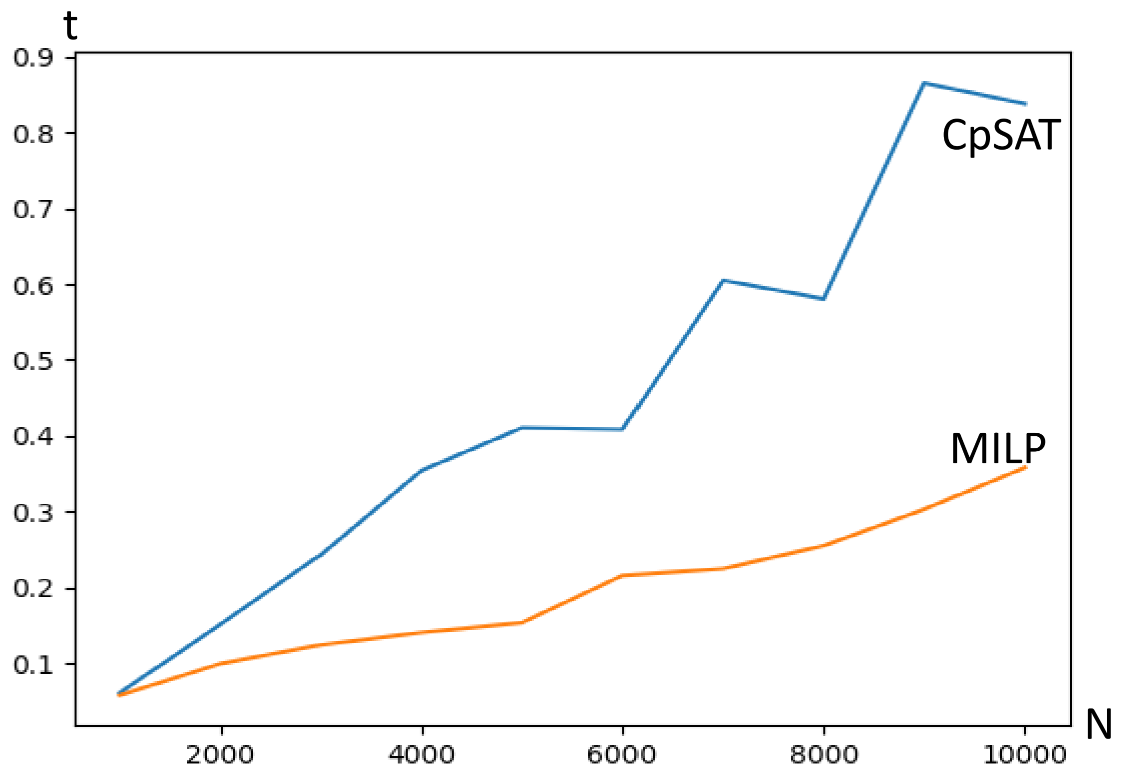
Create Boolean decision variables

Add coefficients to the constraint and the objective function

Solve the MILP

Runtime comparison

- Comparison for N random instances with uniform weights and values, capacity set to 10% of the total weight of items



Thank you for your attention!