Lecture 17: Integer Linear Programming Continued

Sam McCauley

November 8, 2024

Williams College

Admin

- Assignment 3 out; due *Saturday* the 16th (basically: built-in 2 day extension for everyone)
- Some difficult problems; it's OK if you don't get all of them completely correct. Just write what you know.
- Questions?

Survey

- Assignment 3 is your last assignment/homework, we'll just work on the project afterwards
- We'll have some classes for going over previous solutions/doing short project presentations, but there are a few more extra slots
- Two options:
  - The extra class slots will basically be extra office hours where you can work on the project and discuss it with me
  - Normal lectures going over a few cool topics (Burrows-Wheeler Transform, Suffix Trees, Van Emde Boas trees)
- Core question is really: If we have normal lectures on cool (but not easy) topics that are not ever tested, are you interested/will you attend?

Solving ILPs and MIPs

First thought: Can we Use LP Methods?

- *LP relaxation:* just remove the integer constraints

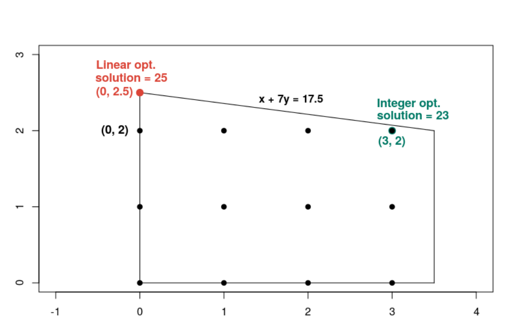
First thought: Can we Use LP Methods?

- *LP relaxation:* just remove the integer constraints
- $e_{i,j} \in \{0, 1\}$ becomes $e_{i,j} \geq 0$ and $e_{i,j} \leq 1$.

First thought: Can we Use LP Methods?

- *LP relaxation:* just remove the integer constraints
- $e_{i,j} \in \{0, 1\}$ becomes $e_{i,j} \geq 0$ and $e_{i,j} \leq 1$.
- How badly can this do?

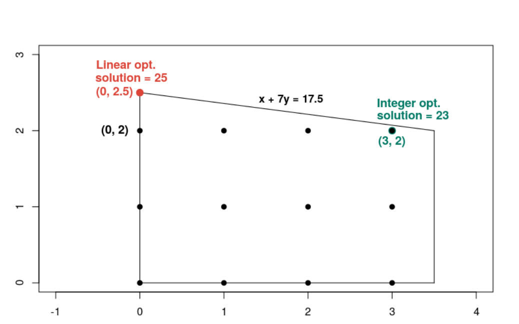
Rounding MIPs



From Google OR Tools Documentation

- Can do *arbitrarily* badly, even for simple ILPs

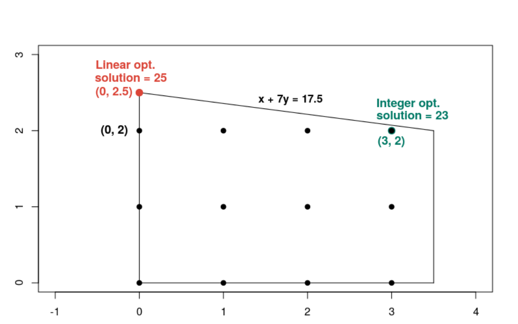
Rounding MIPs



From Google OR Tools Documentation

- Can do *arbitrarily* badly, even for simple ILPs
- May work effectively if the problem has a special structure that makes rounding effective

Rounding MIPs



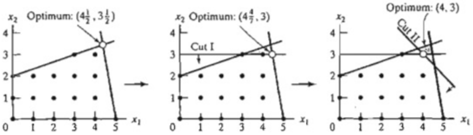
From Google OR Tools Documentation

- Can do *arbitrarily* badly, even for simple ILPs
- May work effectively if the problem has a special structure that makes rounding effective
- Example: the diet problem is probably solved fairly well by rounding (will only be off by 1 unit of each food)

Second Method: Cutting ILPs

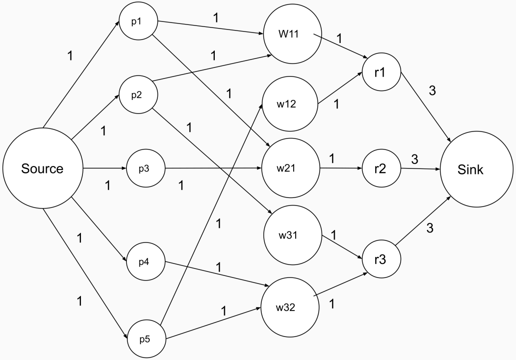
FIGURE 9.10

Illustration of the use of cuts in ILP



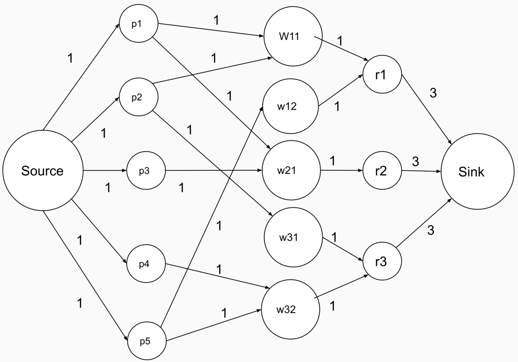
- We *won't cover* in this class
- Cut the LP without removing integer solutions
- After enough cuts, can round and get a good solution!
- Not always possible, but surprisingly effective methods in practice for some types of problem
- Many MIP solvers find these cuts for you

Third Method: Prove the LP has integral soln



- Broad class of LPs are guaranteed to give optimal solutions
- We *won't cover* in this class
- Example for linear algebra people: if your constraint matrix is totally unimodular then there exists an optimal integer solution

Third Method: Prove the LP has integral soln



- Broad class of LPs are guaranteed to give optimal solutions
- We *won't cover* in this class
- Example for flow-reduction-lovers: if you write a flow problems as an LP where all constraints are integers, there exists an optimal integer solution

Main MIP Solving Method: Branch and Bound

Branch and Bound

- Two towers: meet-in-the-middle was faster since we could “rule out” some of the search space

Branch and Bound

- Two towers: meet-in-the-middle was faster since we could “rule out” some of the search space
- Maintain worst-case guarantees

Branch and Bound

- Two towers: meet-in-the-middle was faster since we could “rule out” some of the search space
- Maintain worst-case guarantees
- Branch and bound: a less-problem-specific way to do the same thing

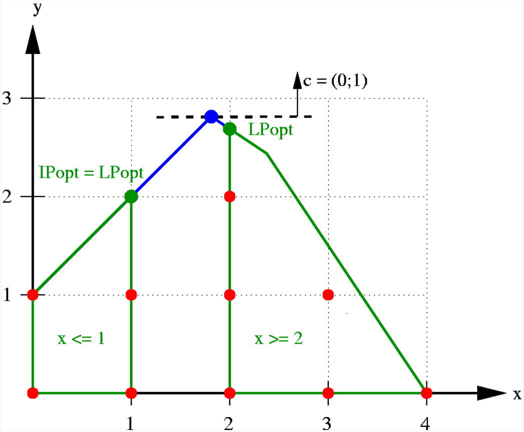
Branch and Bound

- Two towers: meet-in-the-middle was faster since we could “rule out” some of the search space
- Maintain worst-case guarantees
- Branch and bound: a less-problem-specific way to do the same thing
- This is a large *class* of algorithms; I’m giving a high level description of the idea

Branch and Bound

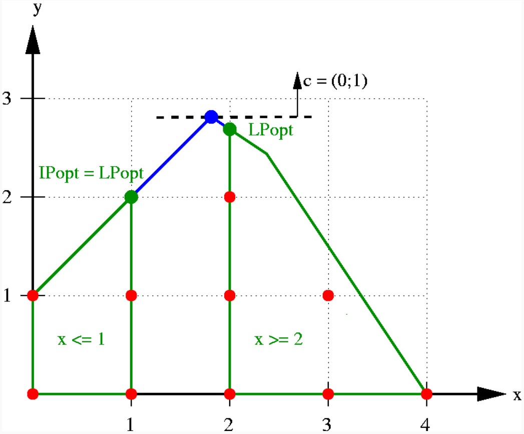
- Two towers: meet-in-the-middle was faster since we could “rule out” some of the search space
- Maintain worst-case guarantees
- Branch and bound: a less-problem-specific way to do the same thing
- This is a large *class* of algorithms; I’m giving a high level description of the idea
- (There is a question about this on Assignment 3.)

Branching



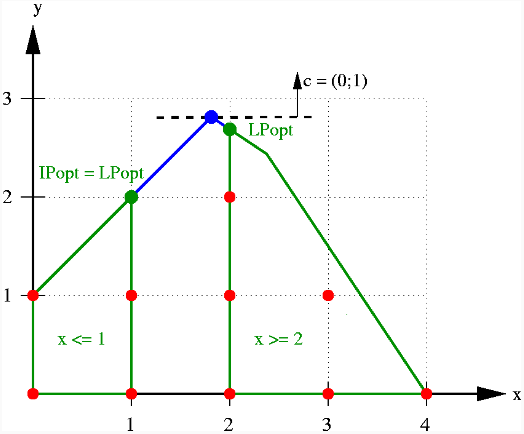
- First, we divide the problem into several subproblems

Branching



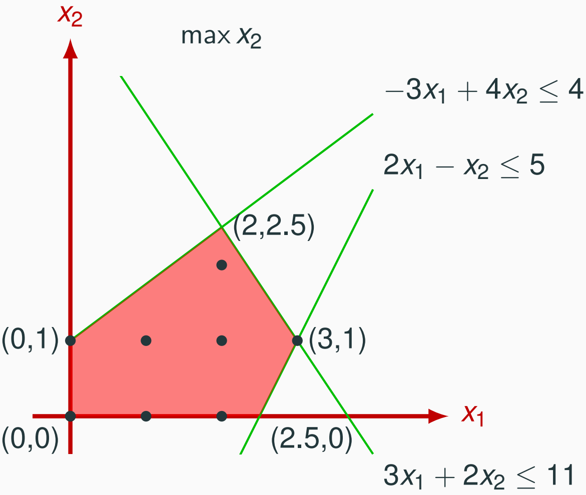
- First, we divide the problem into several subproblems
- Visualization is useful: just partition the feasible region into several pieces

Branching



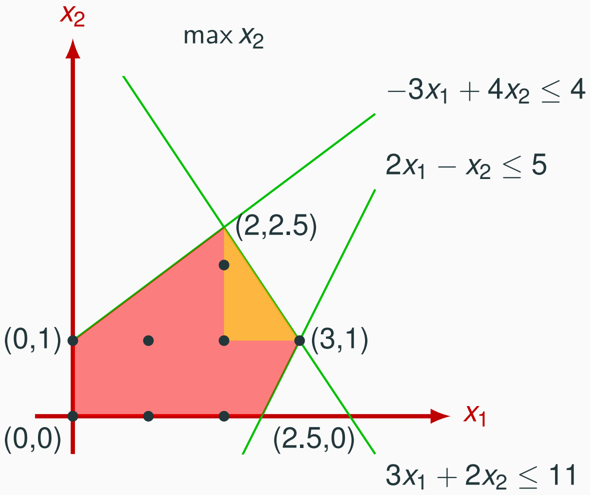
- First, we divide the problem into several subproblems
- Visualization is useful: just partition the feasible region into several pieces
- So far, still need to search through all of them (same as brute force)

Branching and Bounding



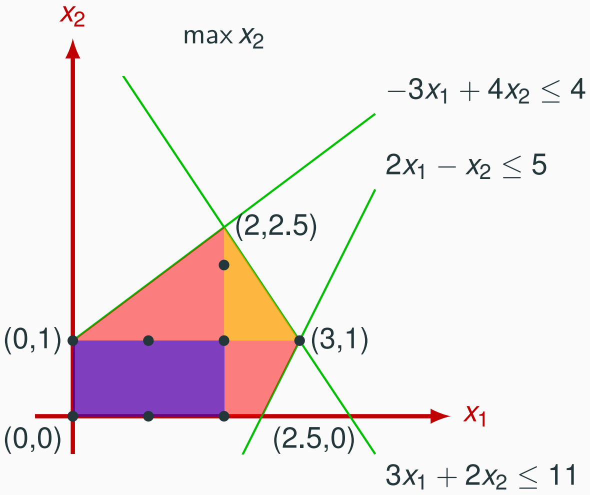
- Partition region

Branching and Bounding



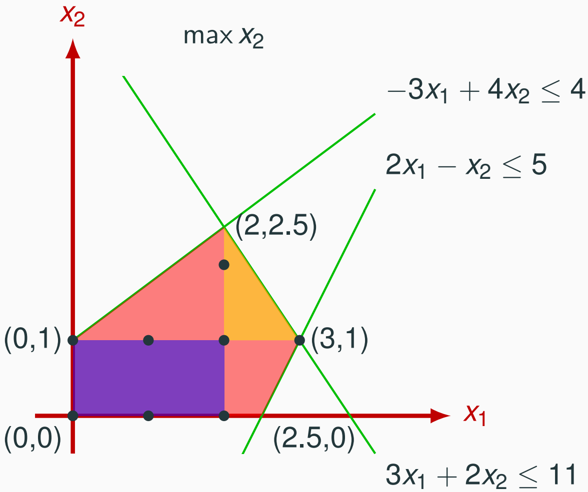
- Partition region
- Find best solution in orange piece

Branching and Bounding



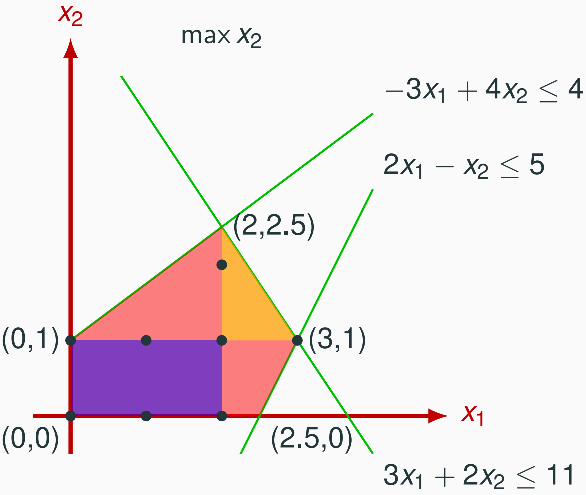
- Partition region
- Find best solution in orange piece
- When can we avoid searching in purple?

Branching and Bounding



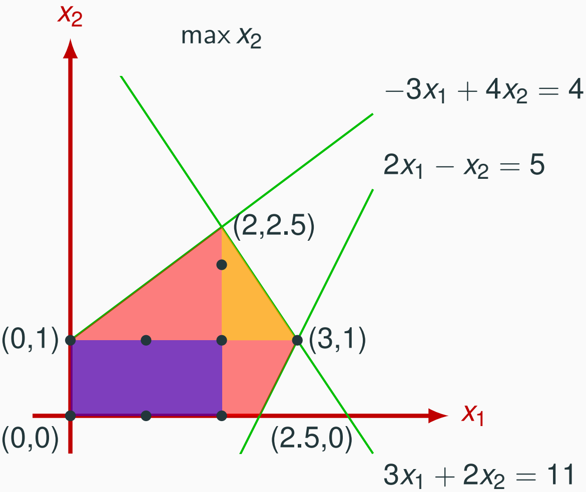
- Upper bound best solution in purple

Branching and Bounding



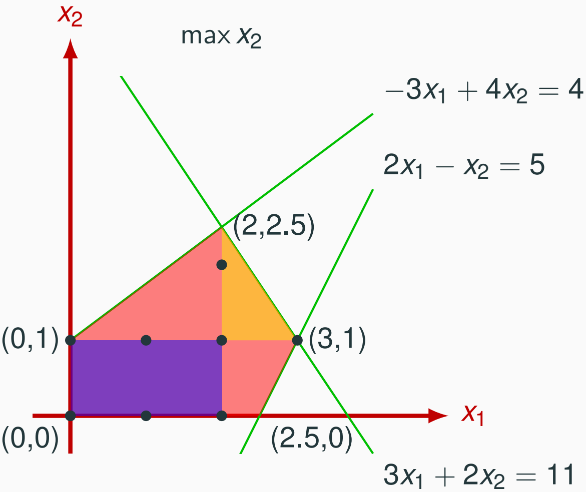
- Upper bound best solution in purple
- If best possible soln in purple is worse than best soln in orange, can *safely skip* it

Branching and Bounding



Safe to skip: *always* still gives an optimal solution.

Branching and Bounding



Safe to skip: *always* still gives an optimal solution.

But, can't skip anything in worst case.

What do we need?

- Way to get a good solution in orange region: recurse!

What do we need?

- Way to get a good solution in orange region: recurse!
- Base case: can just do a simple greedy method if the region is small enough.

What do we need?

- Way to get a good solution in orange region: recurse!
- Base case: can just do a simple greedy method if the region is small enough.
- Way to upper bound best solution in purple region??

What do we need?

- Way to get a good solution in orange region: recurse!
- Base case: can just do a simple greedy method if the region is small enough.
- Way to upper bound best solution in purple region??
  - Relax to an LP! Might not give a good upper bound, but will give *an* upper bound (Recall: LPs are relatively fast to solve)

What do we need?

- Way to get a good solution in orange region: recurse!
- Base case: can just do a simple greedy method if the region is small enough.
- Way to upper bound best solution in purple region??
    - Relax to an LP! Might not give a good upper bound, but will give *an* upper bound (Recall: LPs are relatively fast to solve)
    - Duality can help (we won't talk about in this class)

Branch and Bound Intuition

- Branch: split feasible region into pieces; Bound: bound the solution quality on each so we can rule out searching in some pieces

Branch and Bound Intuition

- Branch: split feasible region into pieces; Bound: bound the solution quality on each so we can rule out searching in some pieces
- Let us rule out big parts of the search space

Branch and Bound Intuition

- Branch: split feasible region into pieces; Bound: bound the solution quality on each so we can rule out searching in some pieces
- Let us rule out big parts of the search space
- “Everything in here has a bad objective function, so we can skip it.” (This is the *bound* part)

Branch and Bound Intuition

- Branch: split feasible region into pieces; Bound: bound the solution quality on each so we can rule out searching in some pieces
- Let us rule out big parts of the search space
- “Everything in here has a bad objective function, so we can skip it.” (This is the *bound* part)
- Many practical problems have large parts that are easy to skip. (If we’re stacking groceries on pallets, no need to spend time looking at solutions with bread on the bottom.)

Branch and Bound Intuition

- Branch: split feasible region into pieces; Bound: bound the solution quality on each so we can rule out searching in some pieces
- Let us rule out big parts of the search space
- “Everything in here has a bad objective function, so we can skip it.” (This is the *bound* part)
- Many practical problems have large parts that are easy to skip. (If we’re stacking groceries on pallets, no need to spend time looking at solutions with bread on the bottom.)
- The more we branch (find good solutions), the more we can bound (rule out parts of the search space whose solutions are suboptimal)

Branch and Bound in Practice

- Advanced methods to figure out how to split into pieces; how much to search each piece before doing more bound calculations

Branch and Bound in Practice

- Advanced methods to figure out how to split into pieces; how much to search each piece before doing more bound calculations
- The better your choices, the more you can rule out

Branch and Bound in Practice

- Advanced methods to figure out how to split into pieces; how much to search each piece before doing more bound calculations
- The better your choices, the more you can rule out
- Other methods (greedy, LP cuts, duality, heuristic search, etc.) can be integrated into this method

Branch and Bound in Practice

- Solvers are sometimes optimized for a given problem

Branch and Bound in Practice

- Solvers are sometimes optimized for a given problem
- Dedicated solvers for TSP, Knapsack, that make branching decisions and use bounding methods particularly effective for that problem

Branch and Bound in Practice

- Solvers are sometimes optimized for a given problem
- Dedicated solvers for TSP, Knapsack, that make branching decisions and use bounding methods particularly effective for that problem
- This is how you get the optimal, giant TSP tours

Branch and Bound in Practice

- Solvers are sometimes optimized for a given problem
- Dedicated solvers for TSP, Knapsack, that make branching decisions and use bounding methods particularly effective for that problem
- This is how you get the optimal, giant TSP tours
- Also some general-purpose solvers

Branch and Bound Summary

- Always gives an optimal solution *eventually*

Branch and Bound Summary

- Always gives an optimal solution *eventually*
- May not find it *quickly* on tricky problems

Branch and Bound Summary

- Always gives an optimal solution *eventually*
- May not find it *quickly* on tricky problems
- Can be very fast even on reasonably hard, reasonably large instances.

Solvers

These solvers have both LP and MIP solvers (using different algorithms):

- GLPK (simplex, branch and bound). Open source. Standalone program is fairly easy to use; can also access from C.
- CPLEX - IBM software for MIPs. Old but reliable. Proprietary. Effective, but can be difficult to work with
- COIN-OR - open source solver
- Google OR tools - wrapper for COIN-OR. Has a really nice TSP and Knapsack solvers. More user friendly

More ILP and MIP Examples

Scheduling

- (Aside: scheduling is a major application of ILPs. Lots of different techniques; this is just one example.)

Scheduling

- (Aside: scheduling is a major application of ILPs. Lots of different techniques; this is just one example.)
- Assign n unit-cost jobs to machines.

Scheduling

- (Aside: scheduling is a major application of ILPs. Lots of different techniques; this is just one example.)
- Assign n unit-cost jobs to machines.
- Each job j_i has a type t_i. Two jobs of the same type cannot be assigned to the same machine.

Scheduling

- (Aside: scheduling is a major application of ILPs. Lots of different techniques; this is just one example.)
- Assign n unit-cost jobs to machines.
- Each job j_i has a type t_i. Two jobs of the same type cannot be assigned to the same machine.
- How can we schedule the jobs with the minimum number of machines?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines

- What variables do we want?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines

- What variables do we want?
- Probably: keep track of what job is assigned to what machine

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines

- What variables do we want?
- Probably: keep track of what job is assigned to what machine
- $s_{i,m} = 1$ if job i is assigned to machine m

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines

- What variables do we want?
- Probably: keep track of what job is assigned to what machine
- $s_{i,m} = 1$ if job i is assigned to machine m
- How many machines do we need?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines

- What variables do we want?
- Probably: keep track of what job is assigned to what machine
- $s_{i,m} = 1$ if job i is assigned to machine m
- How many machines do we need?

- At most n. So have n^2 variables: $s_{i,m} \in \{0, 1\}$, for $1 \leq i \leq n$ and $1 \leq m \leq n$.

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- Want every job assigned to exactly one machine

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- Want every job assigned to exactly one machine
- For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- Two jobs of the same type can't be assigned to the same machine

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- Two jobs of the same type can't be assigned to the same machine
- Rephrased: for every machine m, no two jobs of the same type can be assigned to m

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- For every machine i, no two jobs of the same type can be assigned to i

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- For every machine i, no two jobs of the same type can be assigned to i
- For all $1 \leq m \leq n$,for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Constraints?
- For every machine i, no two jobs of the same type can be assigned to i
- For all $1 \leq m \leq n$,for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$
- (Up to n^3 constraints. Also: constraints depend on the input.)

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Objective?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Objective?
- Let c_m be the cost of machine m. Want $c_m = 1$ if there is a job assigned to machine i, $c_m = 0$ otherwise.

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Objective?
- Let c_m be the cost of machine m. Want $c_m = 1$ if there is a job assigned to machine i, $c_m = 0$ otherwise.
- $\min \sum_{m=1}^{n} c_m$

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Objective?
- Let c_m be the cost of machine m. Want $c_m = 1$ if there is a job assigned to machine i, $c_m = 0$ otherwise.
- $\min \sum_{m=1}^{n} c_m$
- Constraint for c_m?

Scheduling Jobs with Types

- n jobs, job i has type t_i
- Two jobs of same type cannot be assigned to the same machine
- Min number of machines
- $s_{i,m} = 1$ if job i assigned to machine m

- Objective?
- Let c_m be the cost of machine m. Want $c_m = 1$ if there is a job assigned to machine i, $c_m = 0$ otherwise.
- $\min \sum_{m=1}^{n} c_m$
- Constraint for c_m?
- For all jobs i and all machines m, $c_m \geq s_{i,m}$

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints
- We will usually want this to be polynomial in the size of the original problem input

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints
- We will usually want this to be polynomial in the size of the original problem input
- The *computation time* is the time it takes to go from an ILP/MIP recipe to a `.lp` file

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints
- We will usually want this to be polynomial in the size of the original problem input
- The *computation time* is the time it takes to go from an ILP/MIP recipe to a `.lp` file
- In other words: the time to calculate all the *constants*!

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints
- We will usually want this to be polynomial in the size of the original problem input
- The *computation time* is the time it takes to go from an ILP/MIP recipe to a `.lp` file
- In other words: the time to calculate all the *constants*!
- We also want this to be *polynomial* in the size of the original problem input

Size and Computation Time for an ILP/MIP

- The *size* of an ILP/MIP is the number of variables times the number of constraints
- We will usually want this to be polynomial in the size of the original problem input
- The *computation time* is the time it takes to go from an ILP/MIP recipe to a `.lp` file
- In other words: the time to calculate all the *constants*!
- We also want this to be *polynomial* in the size of the original problem input
- I will not ask you to calculate these values. I am going over this because any ILP/MIP you give should have polynomial size and polynomial computation time.

Scheduling Jobs with Types

Objective: $\min \sum_{m=1}^{n} c_m$

Constraints:

For all $1 \leq m \leq n$ and $1 \leq i \leq n$, $c_m \geq s_{i,m}$

For all $1 \leq m \leq n$, for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

$s_{i,m} \in \{0, 1\}$ for all $1 \leq i \leq n$, $1 \leq m \leq n$.

What is the size of this ILP?

Scheduling Jobs with Types

Objective: $\min \sum_{m=1}^{n} c_m$

Constraints:

For all $1 \leq m \leq n$ and $1 \leq i \leq n$, $c_m \geq s_{i,m}$

For all $1 \leq m \leq n$, for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

$s_{i,m} \in \{0, 1\}$ for all $1 \leq i \leq n$, $1 \leq m \leq n$.

What is the size of this ILP?

$n + n^2 = O(n^2)$ variables, at most $n^2 + n^3 + n = O(n^3)$ constraints. Multiplying, total size is $O(n^5)$

So the size is polynomial in n.

Scheduling Jobs with Types

Objective: $\min \sum_{m=1}^{n} c_m$

Constraints:

For all $1 \leq m \leq n$ and $1 \leq i \leq n$, $c_m \geq s_{i,m}$

For all $1 \leq m \leq n$, for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

$s_{i,m} \in \{0, 1\}$ for all $1 \leq i \leq n$, $1 \leq m \leq n$.

Computation time?

Scheduling Jobs with Types

Objective: $\min \sum_{m=1}^{n} c_m$

Constraints:

For all $1 \leq m \leq n$ and $1 \leq i \leq n$, $c_m \geq s_{i,m}$

For all $1 \leq m \leq n$, for all jobs i_1 and i_2 with the same type $t_{i_1} = t_{i_2}$, $s_{i_1,m} + s_{i_2,m} \leq 1$

For all $1 \leq i \leq n$, $\sum_{m=1}^{n} s_{i,m} = 1$

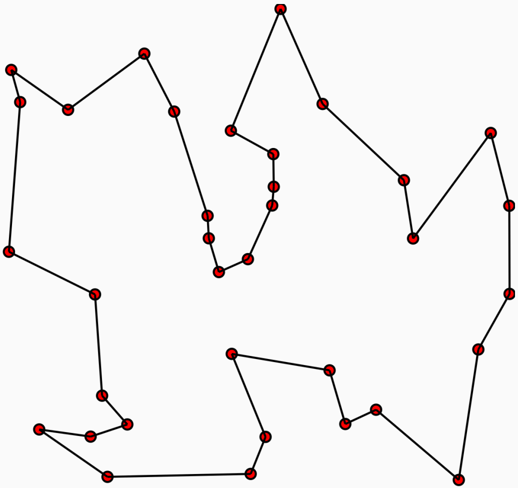
$s_{i,m} \in \{0, 1\}$ for all $1 \leq i \leq n$, $1 \leq m \leq n$.

Computation time?

Polynomial. (All the constants can be calculated in $O(1)$ time.)

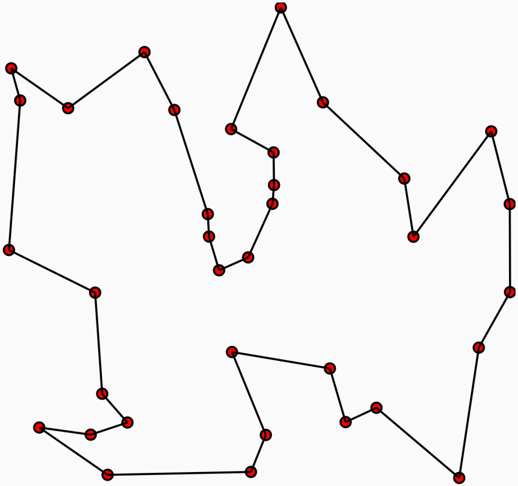
More specifically, this can be caluclated in $O(n^5)$ time.

Travelling Salesman



- Find minimum-length cycle through vertices such that each is visited exactly once

Travelling Salesman



- Find minimum-length cycle through vertices such that each is visited exactly once
- Given: set of n points, for each pair of points i and j the cost $c_{i,j}$ to get from i to j. Have $c_{j,i} = c_{i,j}$

First Attempt: A Solution that Works but is Too Big and Slow

- We want to find the minimum length cycle. Let’s create a variable for every cycle!

First Attempt: A Solution that Works but is Too Big and Slow

- We want to find the minimum length cycle. Let's create a variable for every cycle!
- Let's use variables C_i, and our idea will be that $C_i = 1$ if we use cycle i, $C_i = 0$ otherwise.

First Attempt: A Solution that Works but is Too Big and Slow

- We want to find the minimum length cycle. Let's create a variable for every cycle!
- Let's use variables C_i, and our idea will be that $C_i = 1$ if we use cycle i, $C_i = 0$ otherwise.
- Require: $C_i \in \{0, 1\}$ and $\sum_i C_i = 1$.

First Attempt: A Solution that Works but is Too Big and Slow

- We want to find the minimum length cycle. Let's create a variable for every cycle!
- Let's use variables C_i, and our idea will be that $C_i = 1$ if we use cycle i, $C_i = 0$ otherwise.
- Require: $C_i \in \{0, 1\}$ and $\sum_i C_i = 1$.
- Objective: $\min d_i C_i$ where d_i is the total length of cycle C_i.

Does this work? How big is the LP? How long does it take to calculate?

First Attempt: A Solution that Works but is Too Big and Slow

- We want to find the minimum length cycle. Let's create a variable for every cycle!
- Let's use variables C_i, and our idea will be that $C_i = 1$ if we use cycle i, $C_i = 0$ otherwise.
- Require: $C_i \in \{0, 1\}$ and $\sum_i C_i = 1$.
- Objective: $\min d_i C_i$ where d_i is the total length of cycle C_i.

Does this work? How big is the LP? How long does it take to calculate?

It does work! But the number of variables may be *exponential* in the number of vertices n, and calculating all the d_is also takes (in sum) *exponential* time.

Travelling Salesman: Polynomial Size Solution

- Variables?

Travelling Salesman: Polynomial Size Solution

- Variables?
- $e_{i,j} = 1$ if the TSP tour has an edge from point i to point j

Travelling Salesman: Polynomial Size Solution

- Variables?
- $e_{i,j} = 1$ if the TSP tour has an edge from point i to point j
- $e_{i,j} \in \{0, 1\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$.

Travelling Salesman: Polynomial Size Solution

- Variables?
- $e_{i,j} = 1$ if the TSP tour has an edge from point i to point j
- $e_{i,j} \in \{0, 1\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$.
- Objective?

Travelling Salesman: Polynomial Size Solution

- Variables?
- $e_{i,j} = 1$ if the TSP tour has an edge from point i to point j
- $e_{i,j} \in \{0, 1\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$.
- Objective?
- $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

Travelling Salesman

- Constraints?

Travelling Salesman

- Constraints?
- Need to ensure that the edges with $e_{i,j} = 1$ form a cycle through all points

Travelling Salesman

- Constraints?
- Need to ensure that the edges with $e_{i,j} = 1$ form a cycle through all points
- Observation: in a cycle, all points have one edge coming in, and one edge going out

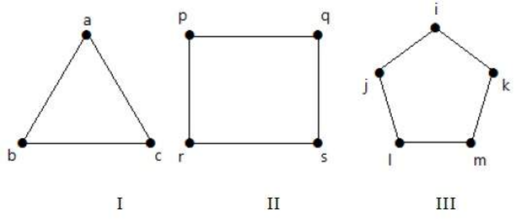
Travelling Salesman

- Constraints?
- Need to ensure that the edges with $e_{i,j} = 1$ form a cycle through all points
- Observation: in a cycle, all points have one edge coming in, and one edge going out
- For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

Travelling Salesman

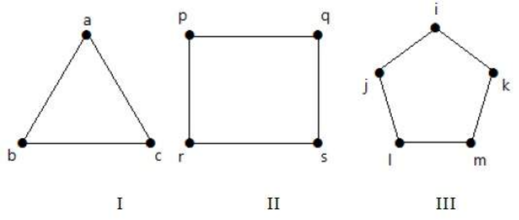
- Constraints?
- Need to ensure that the edges with $e_{i,j} = 1$ form a cycle through all points
- Observation: in a cycle, all points have one edge coming in, and one edge going out
- For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$
- Is this sufficient?

Travelling Salesman



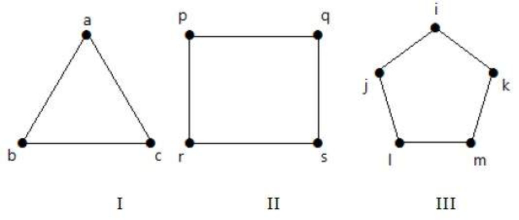
- Unfortunately, no—one in/one out just means a *set* of cycles.

Travelling Salesman



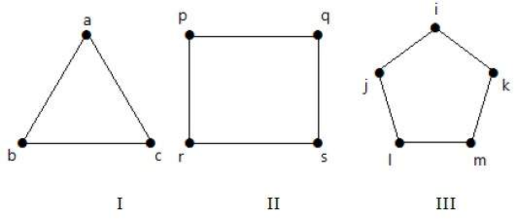
- Unfortunately, no—one in/one out just means a *set* of cycles.
- Can we give another constraint to fix this?

Travelling Salesman



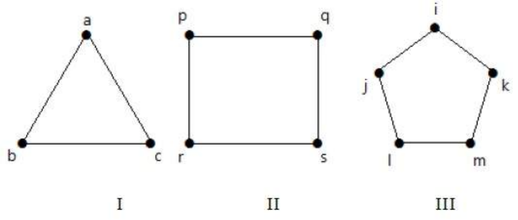
- Unfortunately, no—one in/one out just means a *set* of cycles.
- Can we give another constraint to fix this?
- Somewhat brilliant idea:

Travelling Salesman



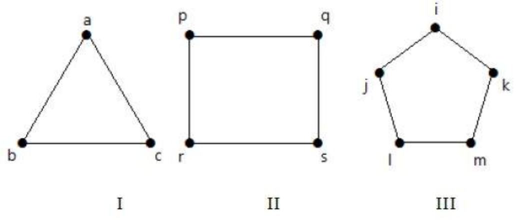
- Unfortunately, no—one in/one out just means a *set* of cycles.
- Can we give another constraint to fix this?
- Somewhat brilliant idea:
- Add $n - 1$ new variables u_i (for $i = 2, \ldots, n$)

Travelling Salesman



- Unfortunately, no—one in/one out just means a *set* of cycles.
- Can we give another constraint to fix this?
- Somewhat brilliant idea:
- Add $n-1$ new variables u_i (for $i=2,\ldots,n$)
- $u_i - u_j + ne_{i,j} \leq n-1$ for $2 \leq i \neq j \leq n$, and

Travelling Salesman



- Unfortunately, no—one in/one out just means a *set* of cycles.
- Can we give another constraint to fix this?
- Somewhat brilliant idea:
- Add $n - 1$ new variables u_i (for $i = 2, \ldots, n$)
- $u_i - u_j + ne_{i,j} \leq n - 1$ for $2 \leq i \neq j \leq n$, and
- $1 \leq u_i \leq n - 1$ for $2 \leq i \leq n$

Travelling Salesman LP

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's prove that this is correct!

Travelling Salesman LP

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's prove that this is correct!
- (Not trivial this time since we have these funny u variables.)

Travelling Salesman LP

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's prove that this is correct!
- (Not trivial this time since we have these funny u variables.)
- Then we'll talk a little bit about intuition.

Travelling Salesman

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- First let's show that if there is a TSP solution C, then there is an LP solution
- If we have a simple cycle visiting every vertex, can we create an assignment that satisfies the constraints?

Travelling Salesman

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- First let's show that if there is a TSP solution C, then there is an LP solution
- If we have a simple cycle visiting every vertex, can we create an assignment that satisfies the constraints?
- Set $e_{i,j} = 1$ if $(i,j) \in C$.

Travelling Salesman

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- First let's show that if there is a TSP solution C, then there is an LP solution
- If we have a simple cycle visiting every vertex, can we create an assignment that satisfies the constraints?
- Set $e_{i,j} = 1$ if $(i, j) \in C$.
- If i is the kth city in C, set $u_i = k$

Travelling Salesman

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- First let's show that if there is a TSP solution C, then there is an LP solution
- If we have a simple cycle visiting every vertex, can we create an assignment that satisfies the constraints?
- Set $e_{i,j} = 1$ if $(i, j) \in C$.
- If i is the kth city in C, set $u_i = k$
- Cost of LP equals cost of C

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's say there is an LP solution. How can we show that it leads to a single cycle?

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's say there is an LP solution. How can we show that it leads to a single cycle?
- Step 1: The first set of constraints means that all edges are a part of some cycle (we'll skip)

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's say there is an LP solution. How can we show that it leads to a single cycle?
- Step 1: The first set of constraints means that all edges are a part of some cycle (we'll skip)
- Step 2: Use the second constraint to show that any cycle C' contains city 1:.

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's say there is an LP solution. How can we show that it leads to a single cycle?
- Step 1: The first set of constraints means that all edges are a part of some cycle (we'll skip)
- Step 2: Use the second constraint to show that any cycle C' contains city 1:.
- Assume by contradiction that C' doesn't contain city 1. Then we'll *sum all* the constraints for edges in C'

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Let's say there is an LP solution. How can we show that it leads to a single cycle?
- Step 1: The first set of constraints means that all edges are a part of some cycle (we'll skip)
- Step 2: Use the second constraint to show that any cycle C' contains city 1:.
- Assume by contradiction that C' doesn't contain city 1. Then we'll *sum all* the constraints for edges in C'
- All the u_i and u_j cancel, and we get $n \leq n - 1$. Since this is impossible, one of the original constraints must not have been satisfied.

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Size of the ILP?

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Size of the ILP?

- At most $n + n^2$ variables; $2n + n^2$ constraints. Total size $O(n^4)$; polynomial!

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Size of the ILP?
- At most $n + n^2$ variables; $2n + n^2$ constraints. Total size $O(n^4)$; polynomial!
- Can we calculate the ILP for an instance in polynomial time?

Travelling Salesman (High Level Proof Idea)

minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} e_{i,j} c_{i,j}$

For all i, $\sum_{j \neq i} e_{i,j} = 1$ and $\sum_{\ell \neq i} e_{\ell,i} = 1$

For all $2 \leq i \neq j \leq n$, $u_i - u_j + n e_{i,j} \leq n - 1$

- Size of the ILP?
- At most $n + n^2$ variables; $2n + n^2$ constraints. Total size $O(n^4)$; polynomial!
- Can we calculate the ILP for an instance in polynomial time?
- Yes, just need to look up the costs $c_{i,j}$.

One last example

- Idea here: we talked about how LPs can only really "AND" constraints

One last example

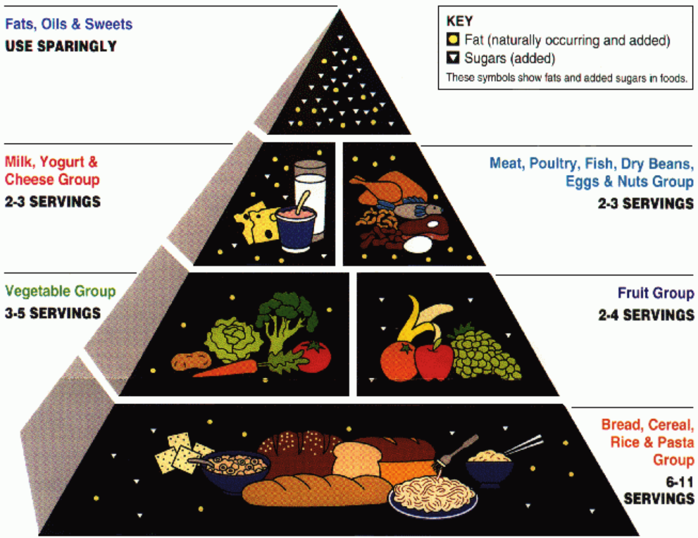
- Idea here: we talked about how LPs can only really “AND” constraints

- With ILP and MIP, can do something much more like “OR”:
  - One of these constraints must be satisfied, or
  - Pick one of these items (in an assignment)

One last example

- Idea here: we talked about how LPs can only really “AND” constraints
- With ILP and MIP, can do something much more like “OR”:
  - One of these constraints must be satisfied, or
  - Pick one of these items (in an assignment)
- Simple example: optimal eating while being able to choose your diet

Food Pyramid



Choice of diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day

Choice of diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- 100g Peanuts: 25.8g of protein, 16.1g carbs, $1.61

Choice of diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day

- 100g Peanuts: 25.8g of protein, 16.1g carbs, $1.61

- 100g Rice: 2.5g protein, 28.7g carbs, $.79

Choice of diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- 100g Peanuts: 25.8g of protein, 16.1g carbs, $1.61
- 100g Rice: 2.5g protein, 28.7g carbs, $.79
- 100g Chicken: 13.5g protein, 0g carbs, $.70

What is the cheapest way you can hit one of these diet goals?

MIP for Choice of Diet

- How to encode which diet I choose?

MIP for Choice of Diet

- How to encode which diet I choose?
- $x_1 = 1$ if I choose the first diet; $x_2 = 1$ if I choosed the second diet; $x_3 = 1$ if I choose the third diet

MIP for Choice of Diet

- How to encode which diet I choose?
- $x_1 = 1$ if I choose the first diet; $x_2 = 1$ if I choosed the second diet; $x_3 = 1$ if I choose the third diet
- Make sure I choose exactly one diet?

MIP for Choice of Diet

- How to encode which diet I choose?
- $x_1 = 1$ if I choose the first diet; $x_2 = 1$ if I choosed the second diet; $x_3 = 1$ if I choose the third diet
- Make sure I choose exactly one diet?
- $x_i \in \{0, 1\}$

MIP for Choice of Diet

- How to encode which diet I choose?
- $x_1 = 1$ if I choose the first diet; $x_2 = 1$ if I choosed the second diet; $x_3 = 1$ if I choose the third diet
- Make sure I choose exactly one diet?
- $x_i \in \{0, 1\}$
- $x_1 + x_2 + x_3 = 1$

MIP for Choice of Diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- How can I encode this?

MIP for Choice of Diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- How can I encode this?
- Previously: $25.8p + 2.5r + 13.5c \geq 46 \ldots$

MIP for Choice of Diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- How can I encode this?
- Previously: $25.8p + 2.5r + 13.5c \geq 46 \ldots$
- Hint: if $x_1 = 0$, I want to do something to these constraint so that they're *always* satisfied

MIP for Choice of Diet

- You need to satisfy one of the three following diet goals:
  - 46 grams of protein and 130 grams of carbs every day; or
  - 20 grams of protein and 200 grams of carbs every day; or
  - 100 grams of protein and 30 grams of carbs every day
- How can I encode this?
- Previously: $25.8p + 2.5r + 13.5c \geq 46 \ldots$
- Hint: if $x_1 = 0$, I want to do something to these constraint so that they're *always* satisfied
- $25.8p + 2.5r + 13.5c + 46(1 - x_1) \geq 46$

Choice of diet LP

- Diet options:
  - 46 g protein; 130 g carbs; or
  - 20 g protein; 200 g carbs; or
  - 100 g protein; 30 g carbs
- 100g Peanuts: 25.8g protein, 16.1g carbs, $1.61
- 100g Rice: 2.5g protein, 28.7g carbs, $.79
- 100g Chicken: 13.5g protein, 0g carbs, $.70

$\min 1.61p + .79r + .7c$

- $25.8p + 2.5r + 13.5c + 46(1 - x_1) \geq 46;$
- $16.1p + 28.7r + 130(1 - x_1) \geq 130$
- $25.8p + 2.5r + 13.5c + 20(1 - x_2) \geq 20;$
- $16.1p + 28.7r + 200(1 - x_2) \geq 200$
- $25.8p + 2.5r + 13.5c + 100(1 - x_3) \geq 100;$
- $16.1p + 28.7r + 30(1 - x_2) \geq 30$
- $x_1 + x_2 + x_3 = 1$
- $p, r, c \geq 0; p, r \in \mathbb{Z}; x_i \in \{0, 1\}$

Technique summary

- When want to choose one of several constraints to satisfy:

Technique summary

- When want to choose one of several constraints to satisfy:
- Multiply the indicator variable for whether or not you choose by a large enough constant to make the constraint trivial

Technique summary

- When want to choose one of several constraints to satisfy:
- Multiply the indicator variable for whether or not you choose by a large enough constant to make the constraint trivial
- Need to be able to bound the constraint to do this!

Technique summary

- When want to choose one of several constraints to satisfy:
- Multiply the indicator variable for whether or not you choose by a large enough constant to make the constraint trivial
- Need to be able to bound the constraint to do this!
- What happens with rounding when you use this technique?

That’s all for today

- Tuesday: A couple more practice problems; might end early

That's all for today

- Tuesday: A couple more practice problems; might end early
- Friday: talk about the final project, review solutions