Project 1: Knapsack and Graph Coloring

Zach Neveu

May 7, 2019

1 Knapsack

Due: Monday 5/13

Given: Knapsack Instance
8 <- # of objects
1639 <- # Size bound
0 22 27 <- number, value, size

- Within given time, find best soln. on your computer
- Write function exhaustiveKnapsack(Knapsack &K, int t)
- Use given code to read from files
- Write code to exhaustively run knapsack search
- Classes for items - use `select` functions to select items
- Use `printSolution()` to dump results of knapsack once optimal found
- Submit `resultsX.txt` with results
- Ok to compare results with other groups
- 20-30 instances to solve, 10 mins per piece
- Knapsack is the easy part, get this done!
- Just use brute force

2 Graph Coloring

- Idea: Map of USA, adjacent states can't be same color. How many colors needed? If colors are fixed, what is lowest # of conflicts?

Analysis, Big O and Growth of Functions

Zach Neveu

May 27, 2019

1 Book Keeping

- Reading posted
- Lab 1 available

2 Analysis of Algorithms

Problem: a general description of input parameters and the properties that an optimal solution should have

Instance: a specific example of a problem with all parameters specified

- Example: Given a weighted graph, find the cheapest Hamiltonian Cycle (TSP)
- A "problem" can have many instances

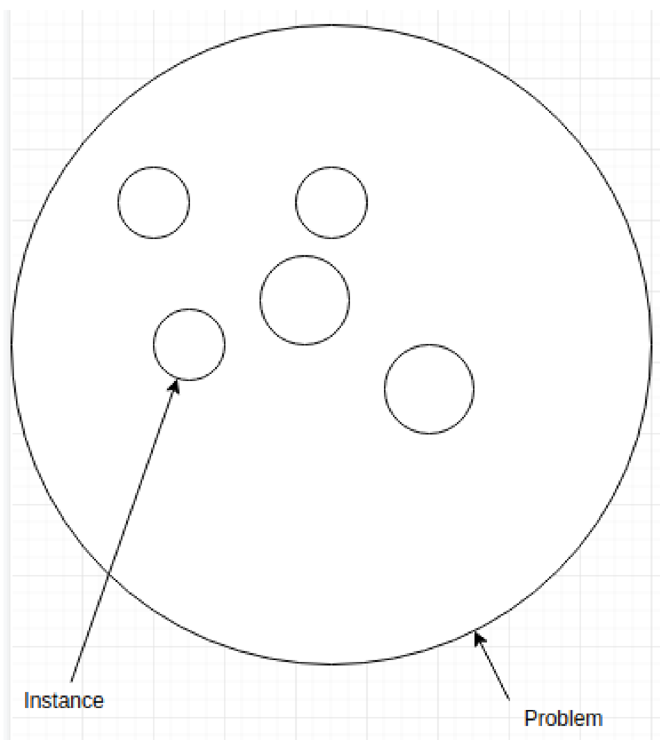

Figure 1: instance_problem

- An algorithm solves all instances of problem

- Many algorithms, what is most efficient?
- What is efficient?
  - Memory
  - Time
  - CPU cycles
  - Disk Space
  - I/O bandwidth
  - Power
- Efficiency usually defined as using smallest time
- Index runtimes by instance size
- "Instance Size" not always well defined - can have multiple params (edges, nodes)

3 Example: Insertion Sort

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
do *key* $\leftarrow A[j]$	c_2	$n-1$
$\triangleright$ Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Figure 2: Cor p.24

- Best case: already sorted. $T(j) = 1, T(n) = an + b \rightarrow$ linear
- Worst case: reverse sorted: $T(j) = j, T(n) = \frac{n(n+1)}{2} \approx an^2 + bn + c \rightarrow$ quadratic

  Time Complexity Function: The largest amount of time for an algorithm needed to solve the problem for a given instance size.
- Even Time-Complexity function considered too complicated for daily use
- Asymptotic notation used instead

4 Asymptotic Notation

For a given function $g(n)$, $O(g(n)) = f(n)$ there exist positive constants k and n_0 such that $f(n) \leq Kg(n)$ for all $n \geq n_0$
Less formally: $O(g(n))$ is the set of functions that are asymptotically less than $g(n)$ for large n.

Example

I claim that $f(n) = an^2 + bn + c = O(n^2)$. If so, then there should exist positive constants k and n_0 such that

$$\begin{gathered} an^2 + bn + c \leq kn^2 \\ a + b/n + \frac{c}{n^2} \leq k \\ k = a + 1 \\ n_0 \; is \; intersection \end{gathered}$$

Summary

- For insertion sort, worst case runtime (time complexity function) is $an^2 + bn + c$ so the complexity is $O(n^2)$

- Also $O(n^3), O(n^4)$ etc.
- Worst case runtime is $O(n^2)$
- Worst case runtime **itself** is upper bound on run time
- $O(n^2)$ is then an upper bound on the general runtime as well!

  Polynomial-time Algorithm: an algorithm whose time complexity function is $O(p(n))$ for some polynomial $p(n)$

  Exponential-time Algorithm: an algorithm that is not polynomial time

EXPONENTIAL VERY BAD

Day 3: Intro to Comp. Complexity, P & NP

Zach Neveu

May 14, 2019

1 Review

- Huge difference between $O(n^k)$ and $O(k^n)$ (Polynomial vs. Exponential)
- Intractable Problems can only be solved (exactly) with an Exponential time algorithm
- Intractable $\neq$ Unsolvable!!

2 New Stuff: Intractable Problems

How to prove a problem is intractable?

1. If solution has size that is exponential, then any algorithm to find that solution can't be polynomial.
2. If problem can't be solved by any algorithm at all (undecidable) - Halting Problem
3. Certain niche problems with small output that are solvable but intractable

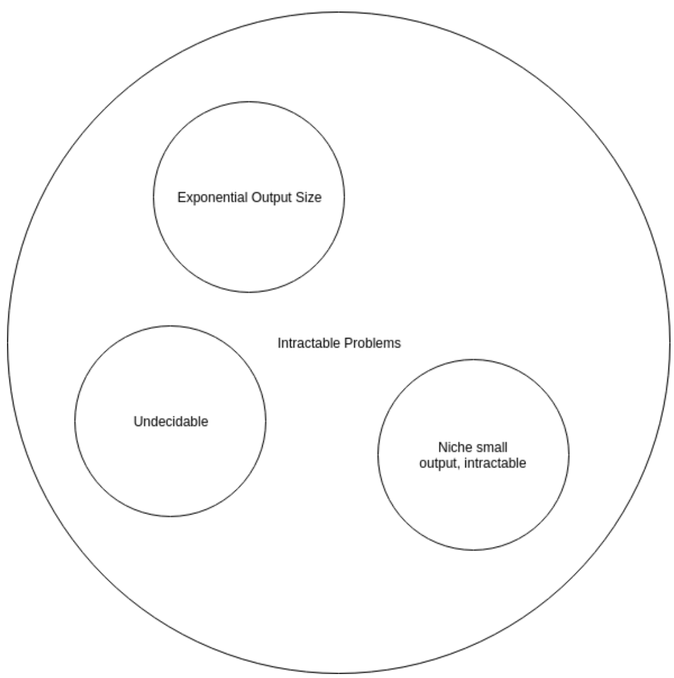

Figure 1: Intractable Problem Categories

3 NP-Completeness

Properties

- No one has ever found a polynomial-time algorithm to solve
- If someone found one algorithm to solve a single NP complete problem in polynomial-time, it would solve all other NP-Complete problems too.
- NP complete problems are either all tractable, or all intractable.
- All in same boat, we just don't know what boat.
- Tons of practical NP complete problems
- Seems unlikely that NP-Complete problems are tractable.
- It is widely believed that the NP-Complete problems are intractable.

What is NP-Complete?

If you can't find an efficient algorithm what do you say?

- "I can't find a solution" - get fired
- "A solution provably doesn't exist" - exceedingly unlikely
- "I can't solve it, but neither can anyone" - show that NP-Complete

Decision vs. Optimization Problems

- Computational Complexity initially developed for decision problems
- Must prove that it can be applied to optimization problems as well
- TSP-opt - Given weighted graph, find shortest Hamiltonian cycle - optimization problem
- TSP-dec - Given weighted graph, is there a Hamiltonian Cycle with weight $\leq k$
- Knapsack-opt - Given objects w/ values and sizes and size bound, find subset to maximize values within size bound.
- Knapsack-dec - Given objects w/ values and sizes and size bound, is there a subset of the objects that are within the size bound and have a value $\geq k$.
- If optimization is tractable, then decision is tractable - just plug in answer from optimization
- If decision is tractable, then optimization is also tractable - just run binary search over k values (adds log of constant)
- Decision and Optimization always have same complexity

4 Complexity Classes

P

- The set of all decision problems that can be solved by a polynomial-time algorithm.
- HC Problem: Given a graph, is it Hamiltonian (does it contain at least one Hamiltonian cycle)?

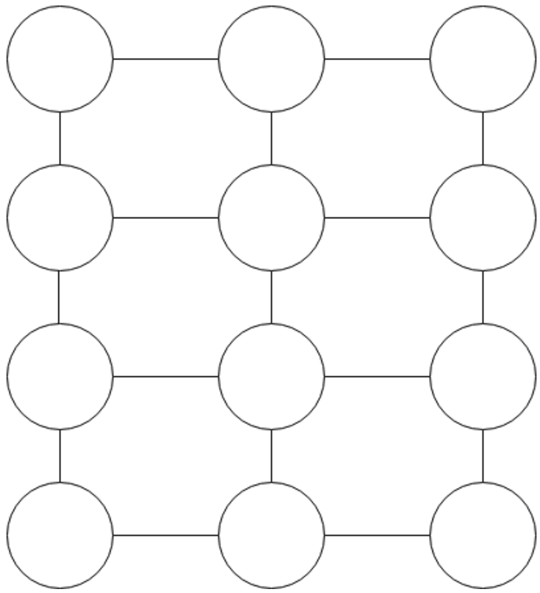

Figure 2: Does this graph have a Hamiltonian Cycle?

- A Verification Algorithm takes some information and checks it to make sure it satisfies the problem requirements.
- Example: given a cycle, verify that it is in fact Hamiltonian
- A **Yes Instance** of a problem will have some certificate that a verification algorithm can verify
- a **No Instance** will not contain a valid certificate to be verified
- Example: TSP-dec
  - certificate: sequence of nodes in the found HC
  - Verification algorithm: Check that each node appears once, edges are valid, and $\sum weights \leq k$
- Example: Matching - given graph and k, is there a matching of size k?
  - Certificate: list of edges in found matching
  - Verification algorithm: make sure no two edges have same end point, edges exist, and number of edges is k
- Shortest Path - given graph, source and dest nodes, and k is there a path from start to end cheaper or equal to k?

- certificate: ordered nodes in the path found
- verification: check that edges exist, $\sum weight \leq k$

NP

- Set of all decision problems such that yes instances have a certificate that can be verified in polynomial-time, and no instances do not have a certificate that can be verified in polynomial-time.
- Basically - a verification algorithm exists that works, and it runs in polynomial-time.
- Verification alg. for HC problem runs in polynomial time for yes instances, so HC $\in$ NP
- TSP: also in NP
- Matching: also in NP

Day 4: Reducibility, NP-Completeness, Key Results

Zach Neveu

May 9, 2019

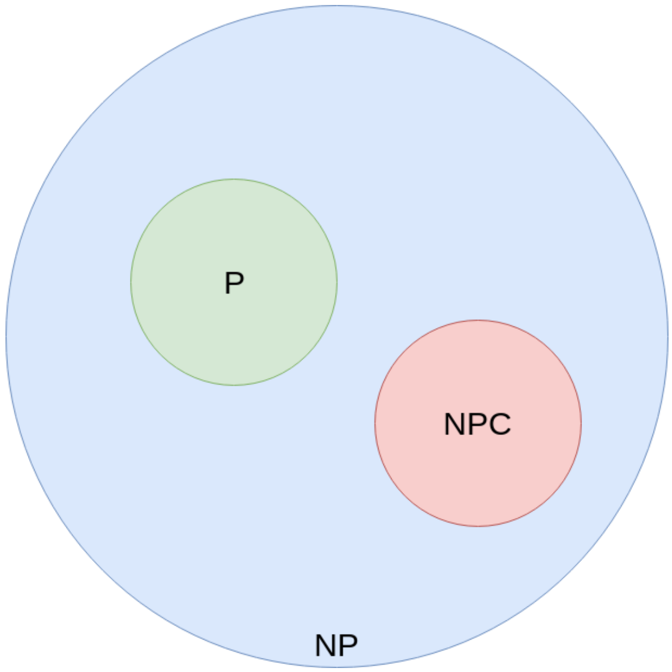

Figure 1: Venn Diagram of P, NP-Complete, and NP

1 Reducibility

Example:
Subset Sum: Given a set of integers and a target, t, is there a subset, S for which $\sum S = t$.
Subset Partition: given a set of integers, can they be partitioned into 2 sets with equal sums?

- If Subset Sum is solved, is it possible to solve subset partition?
- YES! Solve subset sum with $t = \frac{1}{2}\sum S$ where S is all items
- We've just used an SS solver to solve SP! This means that SP reduces to SS.
- If Instance is "no" in SS, it is also "no" in SP

Reducibility: Given problems L_1 and L_2, we say that L_1 is reducible to L_2 in polynomial time if we can rewrite any instance of L_1 as an instance of L_2 such that both instances have the same answer.

Notation: $L_1 \leq L_2$ means that L_1 is reducible to L_2. Starting point, L_1, is on the left. $SP \leq SS$

Example: $HC \leq TSP$
Must be able to rewrite HC as TSP such that they have the same answer.

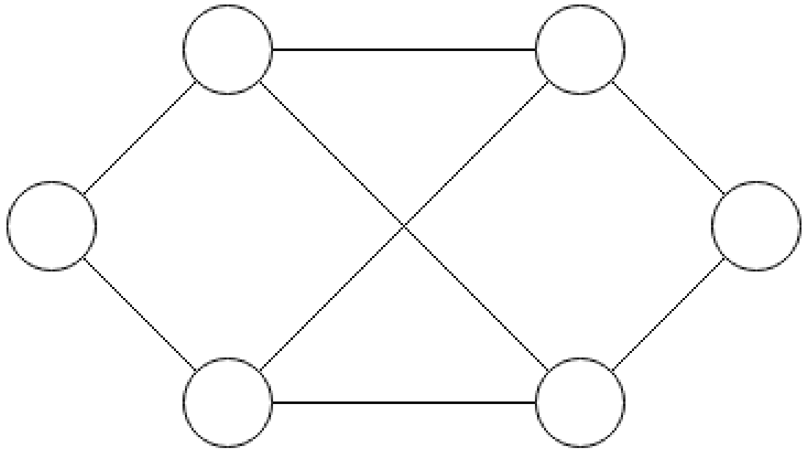

Figure 2: Graph for $HC \leq TSP$ Proof

For proof, must be able to show either:

- A: $yes \rightarrow yes$ and $yes \leftarrow yes$
- B: $yes \rightarrow yes$ and $no \rightarrow no$
- Either A or B requires two steps
- Sometimes one path is much easier
- Option B for $HC \leq TSP$
- If HC is yes instance (HC exists), then the found HC makes TSP a yes instance for weights=1 and bound=num_nodes
- If HC is no instance (no HC exists), then TSP is also no instance because no HCs exist for any cost.

Why is Reduction Useful?

- What if SP is intractable, and SS is in P?
- This is impossible! Reducibility allows you to solve SP in polynomial-time by transforming into SS and solving.

2 NP-completeness

A problem, L, is NP-Complete if:

- $L \in NP$

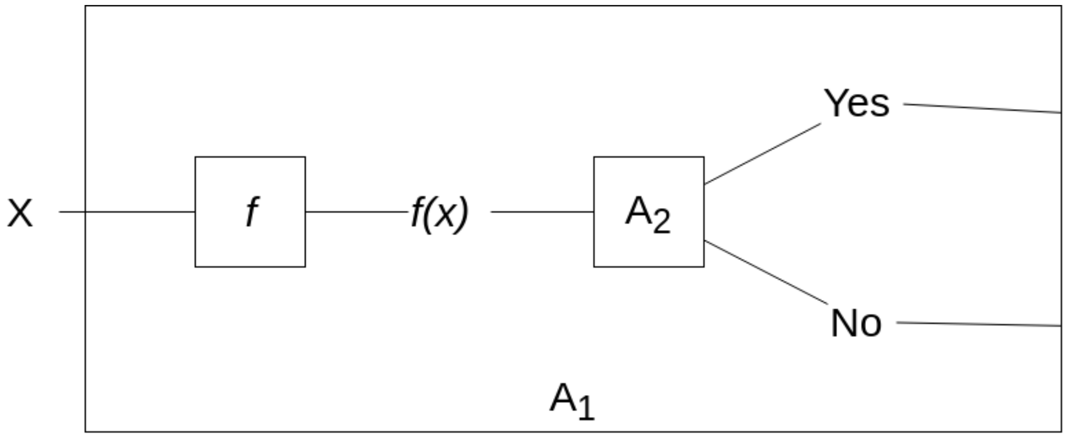

Figure 3: Solving A_1 using A_2 Solver and Reducibility

- For every $L' \in NP$, $L' \leq L$

In words, Every problem in NP should be reducible to L in polynomial time. This essentially means that all NP complete problems are harder than or equal to any other problem in NP. How do we show this?

3 Key Results

1. If $L_1 \leq L_2$, and $L_2 \in P$, then $L_1 \in P$
2. If $L_1 \leq L_2$ and $L_1 \notin P$, then $L_2 \notin P$
3. If L is NPC and $L \in P$, then $NP \in P$
4. If $L' \in NP$ such that $L' \notin P$, then all $NPC \notin P$

4 NPC Examples

Satisfiability (SAT)

- 1971 - Cook found first NPC problem!
- Satisfiability Problem (first one!)
- Consider boolean expression $\overline{x}_3(x_1 + \overline{x}_2 + x_3)$
- Expression is satisfiable if a set of inputs exists which can produce a true output from the expression.
- Given a POS form of an expression, is it satisfiable?
- Ex: $(x_1 + x_2 + x_3)(x_1 + \overline{x}_2)(x_2 + \overline{x}_3)(x_3 + \overline{x}_1)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)$

  - Each clause must be satisfiable
  - Going by hand from left to right, we can find that this isn't satisfiable.
- How can every problem be reduced to this?
- All problems in NP have a verification algorithm
- Verification algorithm can be expressed as a satisfiability instance, this is the reduction.
- This shows that $SAT \in NPC$! First problem ever done.
- This result can be leveraged to prove that other problems are NPC
- $NP \leq SAT$

Evolution of Problems

- Year after SAT, first 10 problems shown to be NPC
- After 50 years there are TONS of problems in the list of NPC
- Problems from every field on here.
- When you have a new problem, look for a similar problem that is proved to be NPC and reduce it to your problem.

Arbitrary Problem L_2

- If $L_1 \in NPC$ and $L_1 \leq L_2$ than $L_2 \in NPC$

Day 5 Notes

Zach Neveu

May 21, 2019

1 Agenda

- Review of NPC
- Proving problems NP Complete
- Examples
- Subproblems

2 Announcements

- Quiz on Wednesday - through most recent homework (NOT NPC)
  - Know big ideas
  - Know important terms
  - Know practical applications
  - Re-Solve problems we've seen for practice
  - Make sure we've done the reading
  - No code on quiz
- Finish up reading about NPC stuff

3 NP Completeness Review

To be in NPC, a problem, L, must:

- $L \in NP$
- For every problem $L' \in NP$, $L' \leq L$

How to Prove

- Prove that $SAT \leq L$
- Given a problem $\pi \in NP$ whos complexity is unknown, how to find?

- Define special case π' containing a subset of the instances of π
- Prove that π' is NPC
- $\pi' \leq \pi$ because every special case is already a regular case as well
- π is NPC, since its simpler subset, π' is NPC
- QUIZ: Explain why the last bullet is true

4 Examples

Partition: given a set **A** and size $s(\boldsymbol{A})$ for all $a \in \boldsymbol{A}$ is there a subset $\boldsymbol{A'} \in \boldsymbol{A}$ such that $\sum s(a) = \sum s(!a)$ where $!a$ is the set of elements not in s. Basically, divide A into two sets with equal size.

Knapsack: given a set, U, a size $s(u)$ and a value $v(u)$ for all $u \in U$, and size constraint B, and a value goal K, is there a subset $u' \in U$ such that $\sum s(u') \leq B$ and $\sum v(u') \geq K$?

- Claim: Partition $\leq$ Knapsack
- Prove: Given an instance of Partition, show that we can produce an instance of knapsack with the same answer.
- Answer: Set $K = B = \frac{1}{2} \sum s(u)$.
- Idea: sandwich K & B such that knapsack will find same answer as partition
- If Knapsack is yes instance, Partition will be yes instance
- If Partition is yes instance, Knapsack is yes instance
- If partition is NPC, Knapsack is NPC

5 Problem as Tuple

- Consider a problem $\pi = (D, Y)$ where D is all instances, Y is all yes instances
- Sub-problem $\pi' = (D', Y')$ reduces to π

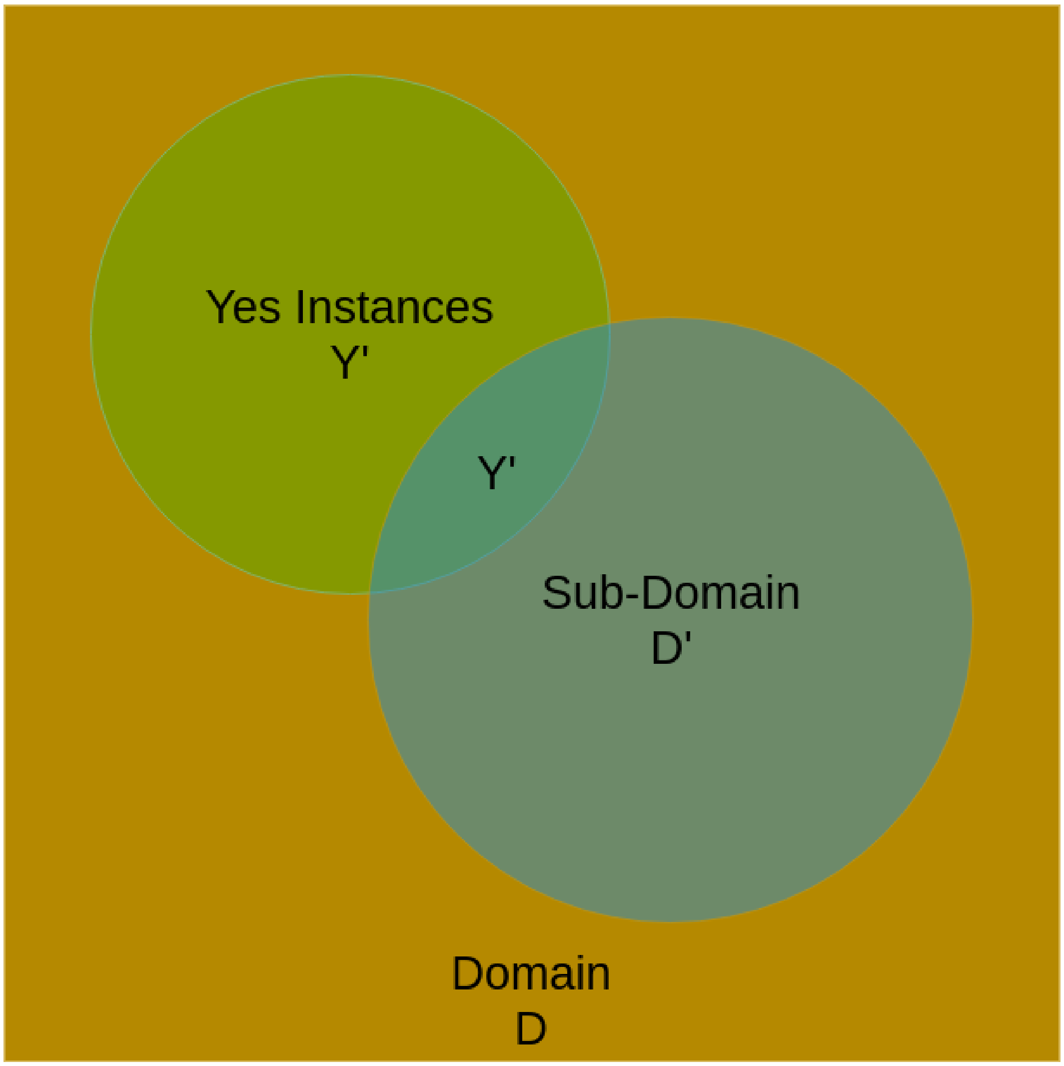

Figure 1: Relation of D, D', Y, Y'

Day 6 Notes

Zach Neveu

May 14, 2019

1 Review

- NPC proof via sub-problem (see Venn diagram from day 5)
- If sub-problem is NPC, problem is NPC
- If problem is P, sub-problem also P
- Sub-problems can be organized recursively (Complexity Landscape$^{\text{TM}}$)

Example: Procedure Constrained Scheduling (PCS)

- Given a set of tasks that each take one unit of time to complete, a partial order on the tasks, a number, m, of processors, and an integer deadline
- Question: Does a legal schedule allow the processes to be completed before the deadline?
- General PCS ∈ NPC
- If constraints graph is tree, $PCS \in P$
- If constraints graph empty, $PCS \in P$
- Aside: Why m=2 solvable, but m=3 so hard?
- Consider: 3 is important.
- $2SAT \in P$, $3SAT \in NPC$, same for HC problem

Special Nodes

A node on the Complexity Landscape that has no known NPC children is called **Minimally NPC**
A node on the Complexity Landscape that is in P and has no parents in P is called **Maximally polynomially solvable**

2 Greedy Algorithms

- A **Greedy Algorithm** always makes what appears to be the best decision in the current moment
- A **Greedy Algorithm** does not utilize backtracking

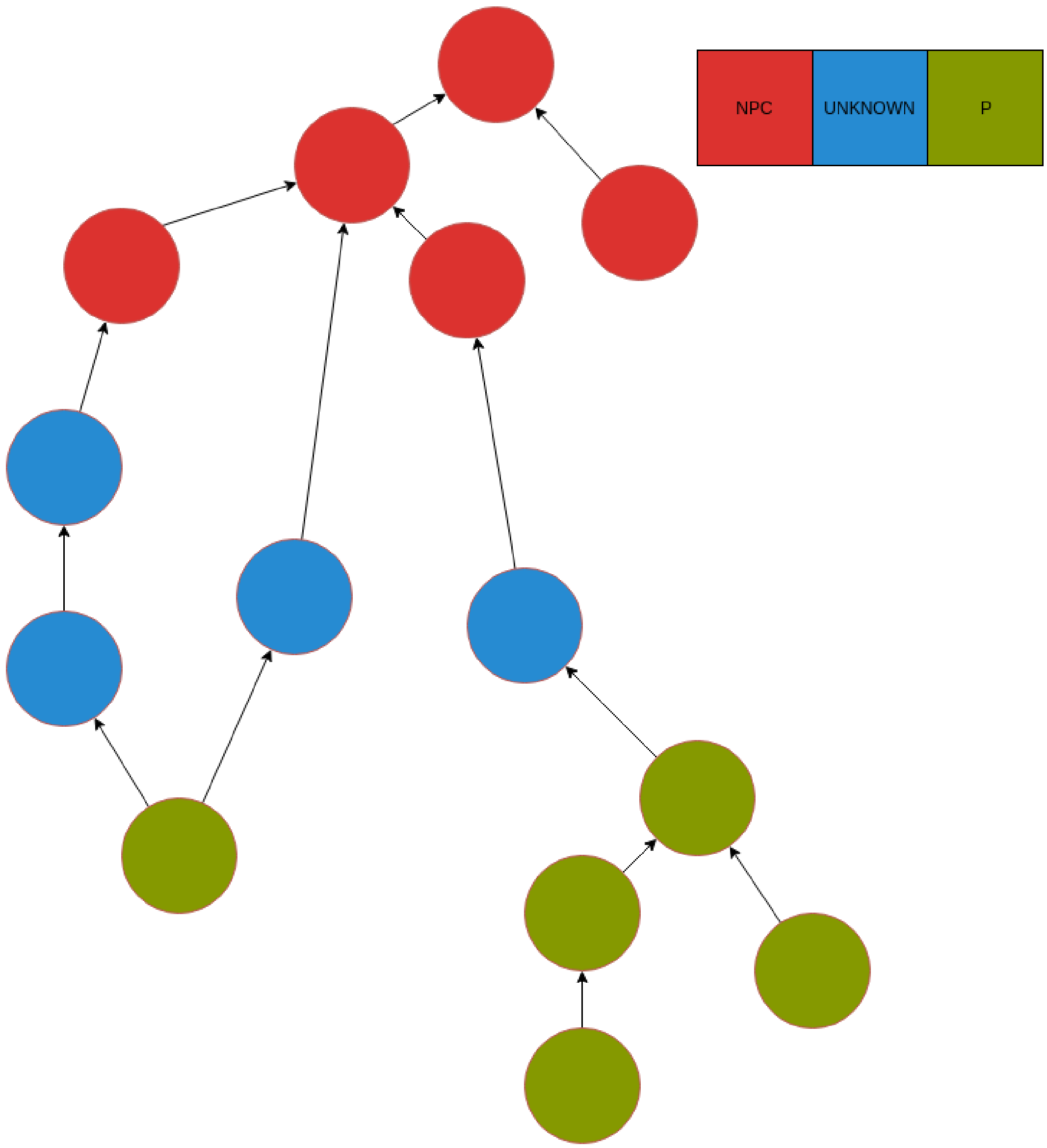

Figure 1: Example Complexity Landscape™

MST

MST: Given an undirected graph and a weight for each edge, find an acyclic subset of the edges that connects all nodes with minimum weight.

```
def MST():
  A=0
  while A is not spanning tree:
    find next edge (u,v) in increasing order by weight such that (u,v) is safe for A
    A += {(u,v)}
  return A
```

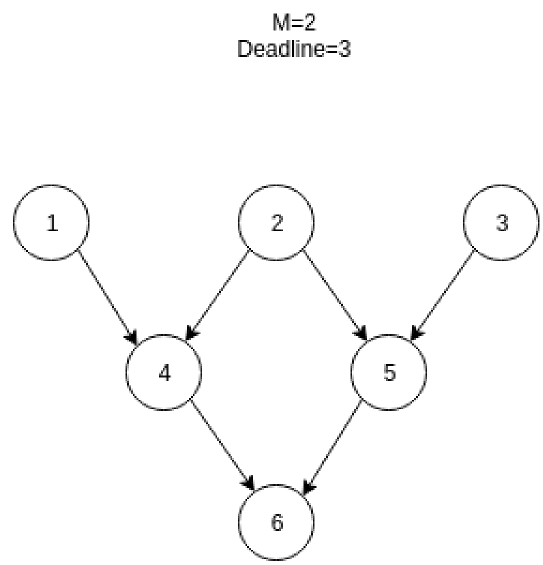

Figure 2: Example PCS Constraint

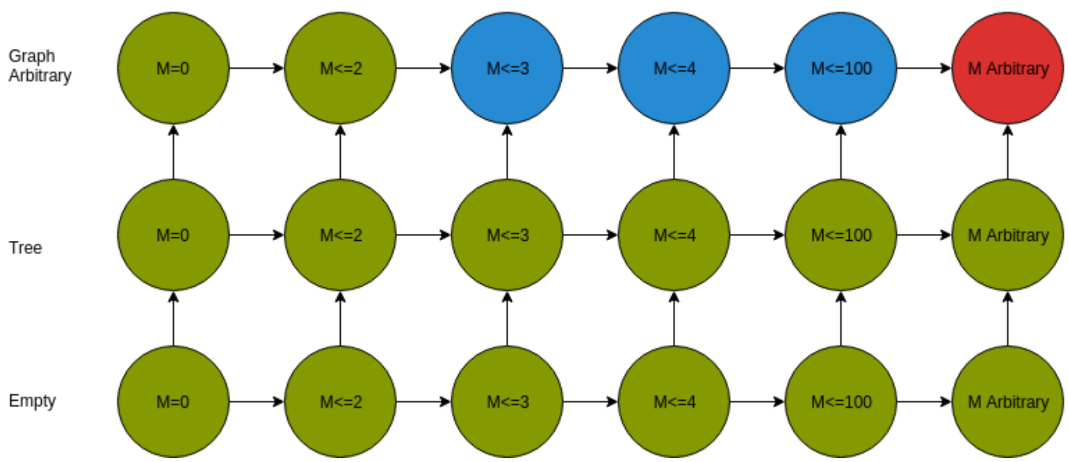

Figure 3: PCS Complexity Landscape™

Activity Selection

- Given: a set $S = \{a_1, a_2, \ldots, a_n\}$ of n activities, only one of which can take place at a time. Activity a_i starts at time s_i and finishes at time f_i. Two activities are **Compatible** if they do not conflict.
- Find: How many activities can we fit?
- Find a largest set of mutually compatible activities

```
# Solution
# a[i] is most recently selected activity
# a[m] activity we are considering adding
def activitySelection(a):
  sortByIncreaseFinish(a)
  n = number of Activities
  A = a[0] # first activity
```

	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	6	8	8	2	12	3	5
f_i	4	5	6	7	10	11	12	13	14	8	9

Table 1: Example Activity Problem

```
i = 1
for m in range(1,n):
  if not conflict(a[m], a[i]):
    A += a[m]
    i = m
return A
```

- Proof: Show that each step is in the right direction
- Prove by contradiction: assume that there is no maximal subset including a[0]. We can always swap the first event in any set with a[0] because of the sorting, so any maximal subset can include a[0].
- Repeat this problem recursively on all problems with $s_i > f_0$ - this step valid for all sub-problems

Head Partition Problem

- Given directed graph, find largest subset of edges which point to separate nodes

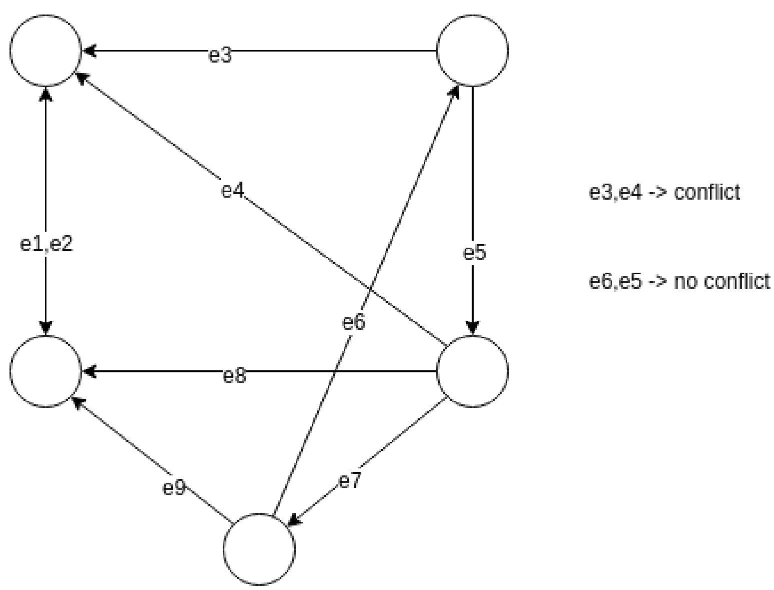

Figure 4: Head Partition Example

```
# Algorithm 1
def headPartition(g):
  for node in g:
    select any incoming arc
```

```python
# Algorithm 2
# Version of "Generic Greedy Algorithm"
def headPartition2(g):
  edge_group = {}
  for arc in g.edges:
    if not conflict(arc, edge_group):
      edge_group += arc
```

Day 7 Notes

Zach Neveu

May 15, 2019

1 Agenda

- Quiz
- Greedy Algorithms
- Intro to matching
- Announce next homework
- Announce next project
- Reading

2 Greedy Algorithms

- Classic Case: Minimum Spanning Tree
- Also useful for many other problems
- Single "greedy algorithm" really at the root of all of them

Head Partition (review from Day 6)

- Node-based solution - finds optimal solution
- Edge-based solulution - finds optimal solution, also very similar to MST
  - Go edge-by-edge, add edge if it does not conflict

Generic Greedy Algorithm

```
def generalGreedy(g):
  sort(g.edges)
  soln = {}
  for edge in g.edges:
    if not conflicts(edge, soln)
    soln += edge
```

Weighted Head Partition

Given a weighted, directed graph, find an independent subset with maximum total weight.

- Edge Strategy: Sort edges by decreasing weight. Starting with highest weight edge, add each edge if it does not conflict, else skip.
- Node Strategy: Go through nodes in any order and select the heaviest edge pointing to the given node

Partition

- Let E be a finite set. π is a partition of E if it is a collection of disjoint subsets of E such that the subsets collectively cover E.
- $E = \{e_1, \ldots, e_8\}$, $\pi = \big\{\{e_1\}, \{e_2, e_3\}, \{e_4, e_5\}, \{e_6, e_7, e_8\}\big\}$
- $\pi.weights = \big\{\{5\}, \{3, 4\}, \{7, 8\}, \{2, 6, 1\}\big\}$
- A subset of E is **Independent** if no two elements come from the same component of π
- $\{e_1, e_4\}$ and $\{e_1, e_4, e_3\}$ are independent
- $\{e_1, e_2, e_3, e_4\}$ is not independent.
- Goal: given E and π, find an independent subset of maximum weight.
- Strategy 1: from each component, select the largest element.
- Same as node strategy for head partition!

Maximum Weighted Matching

- Given a weighted graph, find a matching with the maximum total
- Simple greedy algorithm doesn't work!
- Simple algorithm will choose b,d, a,c, ignoring a,b, c,d

Matching

- Given: a weighted graph and a matching M on the graph
- Edges in M are **matched edges**
- Edges not in M are **free edges**
- Nodes not adjacent to matched edges are **exposed**
- Nodes that are adjacent to matched edges are **matched**

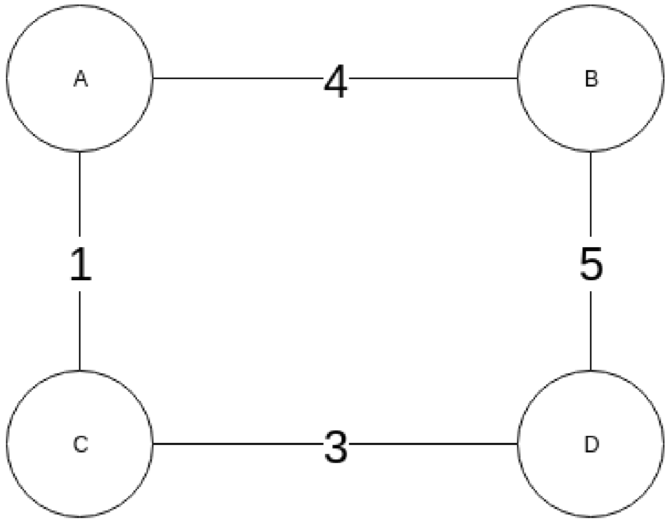

Figure 1: Weighted Matching Example

- **Augmenting Path**: a simple path in the graph beginning and ending at exposed nodes, and alternating between crossing matched and free edges.
- Augmenting Path example in 2: $(v_1, v_4, v_5, v_6, v_8, v_7, v_{10}, v_9)$
- Augmenting Path length always odd because start and end must be on exposed nodes.
- Finding longest augmenting path same as finding largest matching
- Given an augmenting path, swapping edges membership in M along the path creates a valid matching with length one longer.

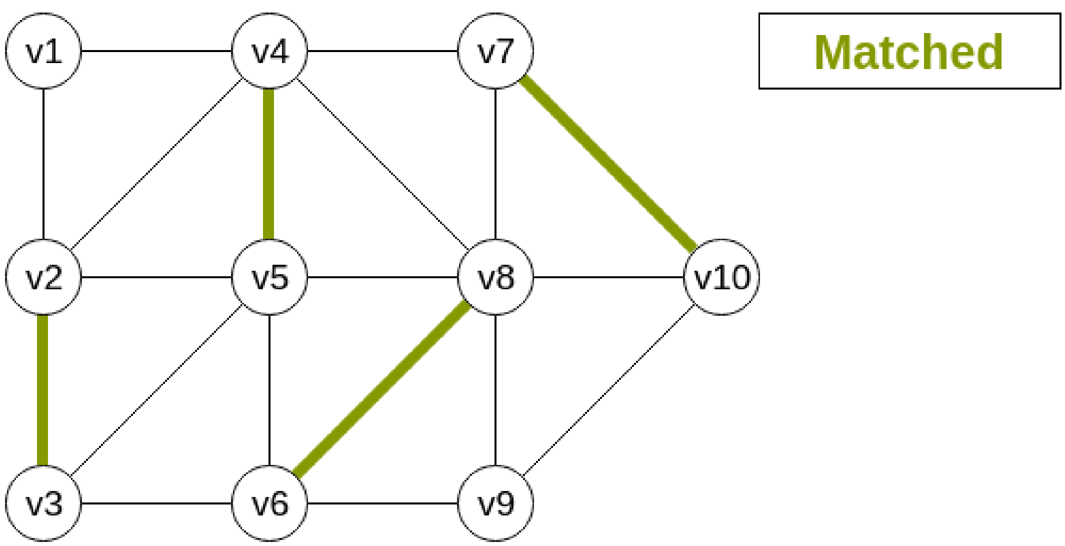

Figure 2: Matching Example

Day 8 Notes

Zach Neveu

May 16, 2019

1 Agenda

- Matching concepts
- Algorithm to solve matching
- Bipartite matching
- Intro to linear programming

2 Matching Review

- Review of matching problem from last class
- Swapping membership of augmenting path yields larger matching
- Maximum matching includes all nodes
- Key result: given a matching, M, the matching is optimal if and only if no augmenting path with respect to M exists.
- If no augmenting path → M is optimal (not so obvious)
- If M optimal → no augmenting path (this is fairly straightforward)

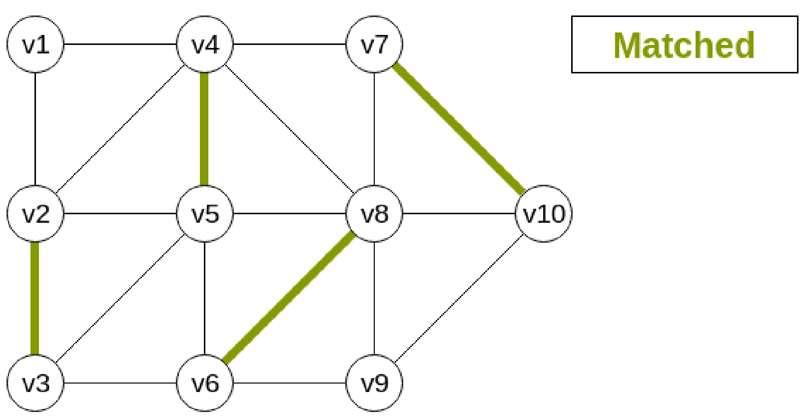

Figure 1: Review matching diagram from Day 6

Matching Algorithm

```
def match(g):
    m = 0
    while p = aug_path(M):
        swap_membership(p)
```

- Outer loop runs $O(E)$ times where E is edges
- Time to find an augmenting path is polynomial time
- Algorithm to find augmenting path is convoluted and not particularly useful to know

Bipartite Matching

- **Bipartite Graph**: a graph where the nodes can be divided into two groups such that every edge goes from one group to the other (no edges are inside a group).
- Bipartite Matching: find the largest matching in a bipartite graph
- Classic example problem: Job scheduling on heterogeneous computers
  - Group of jobs that all need to be done
  - Group of computers that can run jobs
  - Certain jobs can only run on certain computers
  - Find how to get the most jobs done

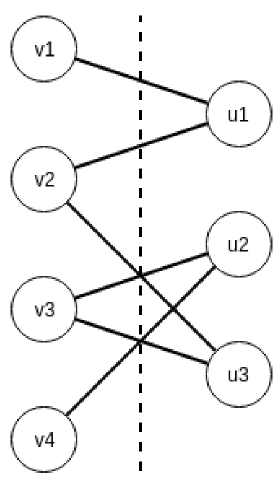

Figure 2: Bipartite Graph Example: Every edge crosses dividing line

- Start at all exposed nodes
- Do BFS across alternating edges until another exposed node is reached
- The path between exposed nodes that is found is an augmenting path
- BFS runs in fast polynomial time and fins any augmenting paths that exist

- Search diagram has structure: matched stages don't branch, matched and unmatched stages alternate
- Why is this like greedy algorithms?
  - Matching always getting bigger
  - Not quite greedy: some edges get deleted when swapping membership
  - Kind of a "deeper" greedy algorithm
  - Approach applicable to many problems

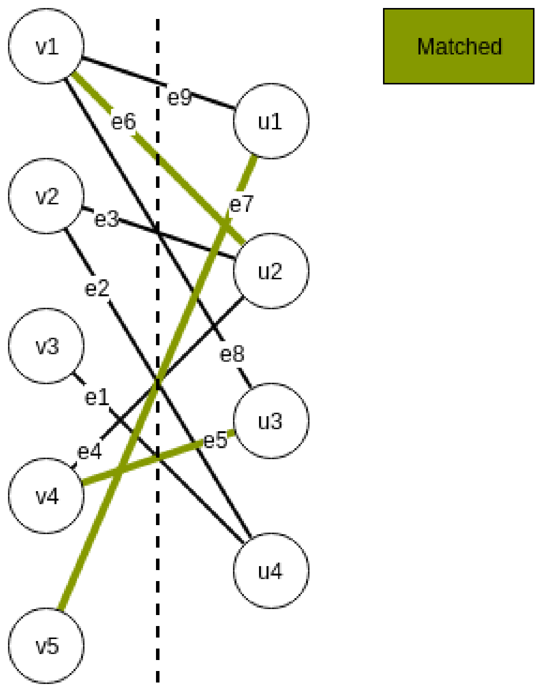

Figure 3: Exaple Bipartite matching

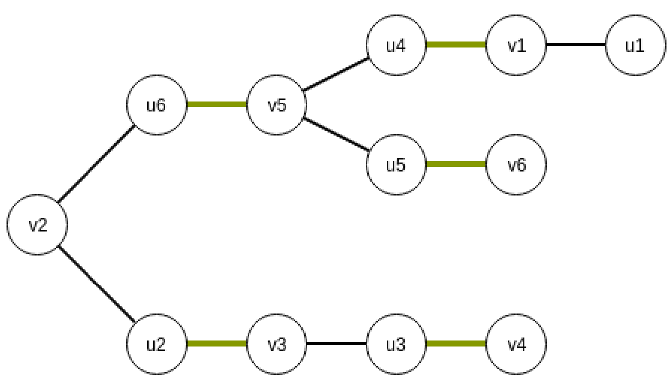

Figure 4: Example search diagram (different problem from fig. 3)

3 Linear Integer Programming (LP)

- Many problems have special format
- If your problem can be re-phrased into this format, it can be solved FAST by existing solvers.
- Leverage genius that is not yours
- Not super well known in ECE, came from applied math
- Large LP instances solvable in minutes
- Trade-off is that problem must be in exact format

Example

- You are a politician trying to win an election. Your district has 3 regions:
  - Urban - 100k voters
  - Suburban - 200k voters
  - Rural - 50k voters
- Goal is to get a majority in each region
- Win votes by advertising based on 4 issues
  - Building roads
  - Gun control
  - Farm subsidies
  - Gas taxes
- Goal: get max results within advertising budget

Table 1: Problem Breakdown Table

	urban	suburban	rural
Build roads	0.2	5	3
gun control	8	2	-5
farm subsidies	0	0	10
gas taxes	10	0	-2

Day 9 Notes

Zach Neveu

May 20, 2019

1 Agenda

- Quizzes back
- No simple mapping from # to grade - based on "expected number"
- Intro to LP
- Examples of LP

2 HW #2

- Definition of NP: **yes** example has certificate which can be verified in polynomial time.
- Definition is not symmetrical!
- Many problems in NP have inverse outside of NP

3 Linear Programming

- Example: Political Election categories (see day 8)
- Ex. Input: \$20k on roads, \$0 on guns, \$4k on farms, \$9k on gas
- Minimize $x_1 + x_2 + x_3 + x_4$ (total cost)

$$20(-2)+0(5)+4(0)+9(10) = 50{,}000$$
$$20(5)+0(2)+4(0)+9(0) = 100{,}000$$
$$20(3)+0(-5)+4(0)+9(-2) = 200{,}000$$

General Linear Program

Given a set of constants $a_1, \ldots, a_n$ and a set of variables $x_1, \ldots, x_n$, a **linear function** f on the variables is:

$$f(x_1, \ldots, x_n) = a_1(x_1) + \ldots + a_n(x_n) = \sum_{\alpha=1}^{n} a_\alpha x_\alpha$$

Given a constant b and a linear function f,
$f() \leq b$ and $f() \geq b$ are linear inequalities
$f() = b$ is a linear equality

$<$ and $>$ are not linear!

LP Problem: The problem of maximizing or minimizing a linear function subject to a finite set of linear constraints.
Example:
Maximize: $x_1 + x_2$
Subject to:
$4x_1 - x_2 \leq 8$
$2x_1 + x_2 \leq 10$
$5x_1 - 2x_2 \geq -2$
$x_1, x_2 \geq 0$

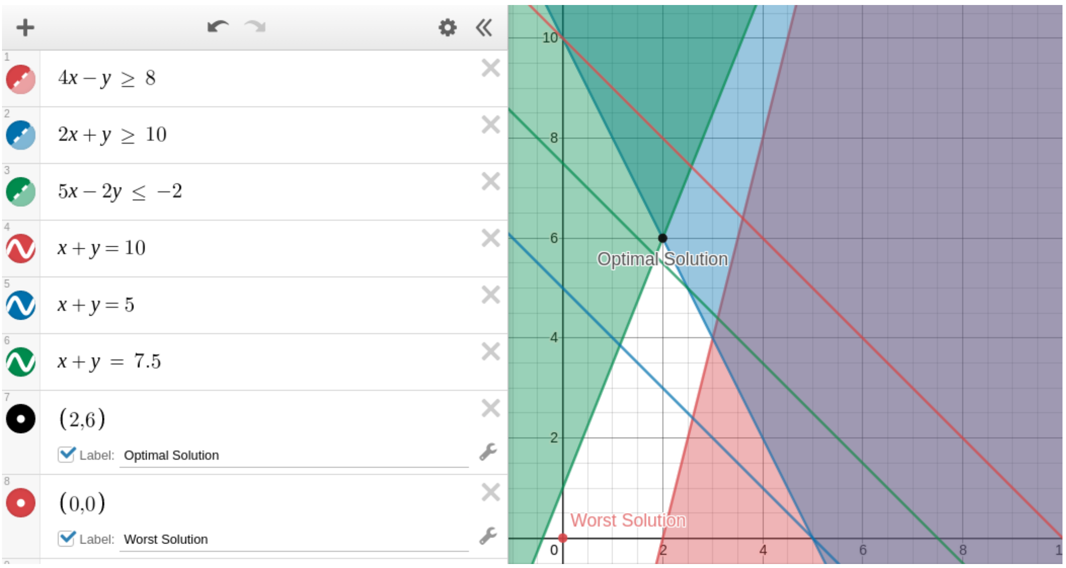

Figure 1: Graph of Solution Space: where all regions overlap is valid (marked in white). $x = y$ lines show value -> further from origin is better.

Example: Reclaiming Solid Waste

Inputs are 3 types of materials: 1,2&3

Table 1: Material Availability

Material	Available Pounds/Week
1	100
2	200
3	300

Problem:
Maximize: $5Y_A + 10Y_B$

Table 2: Grades

Grade	Spec	profit/pound
A	$M_1 \leq 30\%, M_2 \leq 40\%$	5
B	$M_2 = 50\%, M_2 \leq 20\%$	10

Let: Z_{MN} = the proportion of grade M that is material N
Subject to:
$Z_{A_1} \leq 0.3$
$Z_{A_2} \leq 0.4$
$Z_{B_2} = 0.5$
$Z_{B_3} \leq 0.2$
$Y_A * Z_{A_1} + Y_B * Z_{B_1} \leq 100$
$\vdots$
UH OH! This multiplies variables which isn't linear…
Fix:
Let: X_{MN} = <u>amount</u> of Material N in grade M
New Constraints:
$X_{A_1} + X_{B_1} \leq 100$
$X_{A_2} + X_{B_2} \leq 200$
$X_{A_3} + X_{B_3} \leq 300$
New Objective Function:
Maximize: $5(X_{A_1} + X_{A_2} + X_{A_3}) + 10(X_{B_1} + X_{B_2} + X_{B_3})$

Takeaways

- Variable choices matter! Effects speed and solve-ability
- Seemingly non-linear variables can be re-written as linear variables

Standard Form

Given constants $c_1, \ldots, c_n$, $b_1, \ldots, b_m$ and $m * n$ values a_g for $i = 1 \to m$ and $j = 1 \to n$, find $x_1, \ldots, x_n$ such that:
Maximize: $\sum_{j=1}^{n} c_j x_j$
Subject to: $\sum_{i=1}^{n} a_{ij} x_j \leq b_i$ for $i = 1 \to m$
$x_j \geq 0$ for $j = 1 \to n$
n = # variables m = # constraints

- In standard form, all variables $x \geq 0$
- Only maximization as operation
-

Concise Standard Form

- $\boldsymbol{A} = (a_{ij})$
- $\boldsymbol{b} = (b_i)$
- $\boldsymbol{c} = (c_j)$
- $\boldsymbol{x} = (x_j)$

An LP formulation in standard form is: maximize $C^T x$ subject to $Ax \leq b$, $x \geq 0$.

Day 10 Notes

Zach Neveu

May 21, 2019

1 Agenda

- Quiz
- LP Standard Form
- More LP examples
- Solving LP Problems

2 LP Examples

Personnel Scheduling Problem

- Goal: find an assignment of people to shifts such that enough people are working in each slot and total cost is minimized.
- Objective function: $z = 170x_1 + 160x_2 + 175x_3 + 180x_4 + 195x_5$
- Constraint example: $x_1 + x_2 \geq 79$
- Constraint form: $\sum$ active shifts during period $\geq$ required personnel during period.
- Following constraints for all time periods create many redundant constraints
- **redundant constraint:** Constraint can be deleted without changing the solution to the LP
- Solution to this problem: x=(48,31,39,43,15), z=30,610

Table 1: Personnel Schedules

Shift	Time Range	Hourly Wage $
1	6a-2p	170
2	8a-4p	160
3	12p-8p	175
4	4p-12a	180
5	10p-6a	195

3 LP Standard Form

- Maximize $\sum_{j=1}^{n} c_j x_j$ subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1 : m, x_j \geq 0$

Table 2: People Required

Shift	People Needed
6-8	48
8-10	79
10-12	65
12-2	87
2-4	64
4-6	73
6-8	82
8-10	43
10-12	52
12-6a	15

- Always maximizing
- Constraints always use $\leq$
- All variables ≥ 0
- Any LP problem can be rephrased into a standard form equivalent
- **Equivalent:** Two LP formulations are equivalent if they are both maximizing and for every feasible solution in L there is a corresponding feasible solution in L' with the same objective value and visa versa.
- If L is minimizing and L' is maximizing, multiply objective function by -1
- If not all variables are constrained by $\geq 0:$ create 2 positive variables and take their difference to get the unbounded value (e.g. $x_2 \rightarrow (x_2' - x_2'')$)
- If equality used instead of inequality in constraint: Split into $\leq$ and $\geq$ constraints, and flip $\geq$ one.

Day 11 Notes

Zach Neveu

May 22, 2019

1 Agenda

- LP solving with simplex
- Introduction to LP
- Homework #3 due Friday
- Quiz #3 next Tuesday

2 LP

Options for LP soln:

- Single possible solution
- No feasible solution
- Unbounded feasible region
- Infinite optimal solutions along line segment

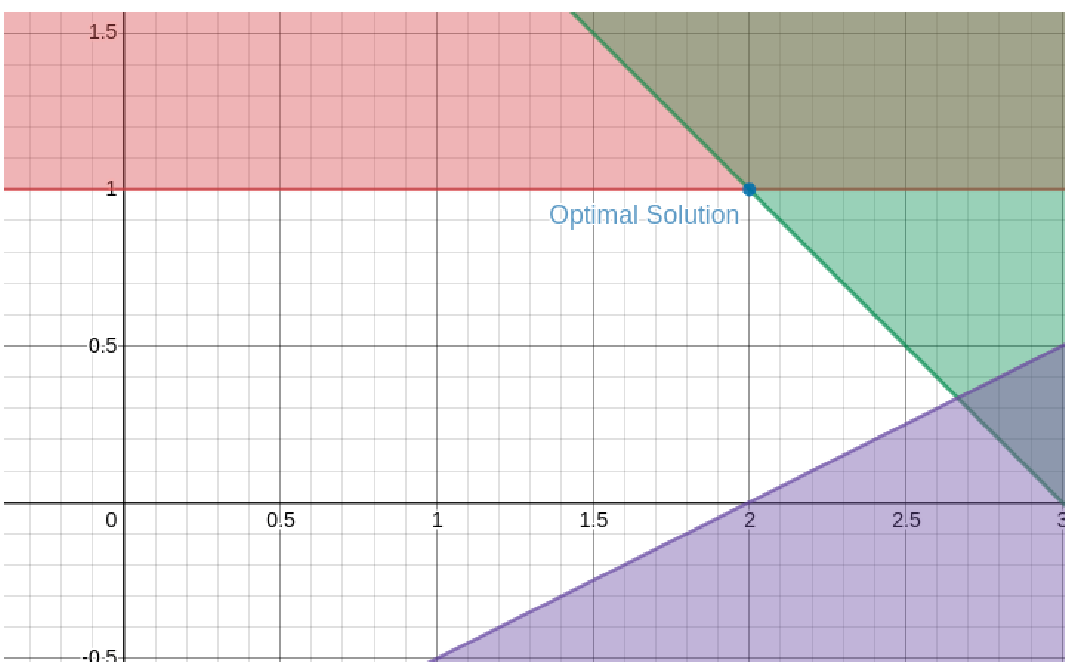

Figure 1: Single Solution

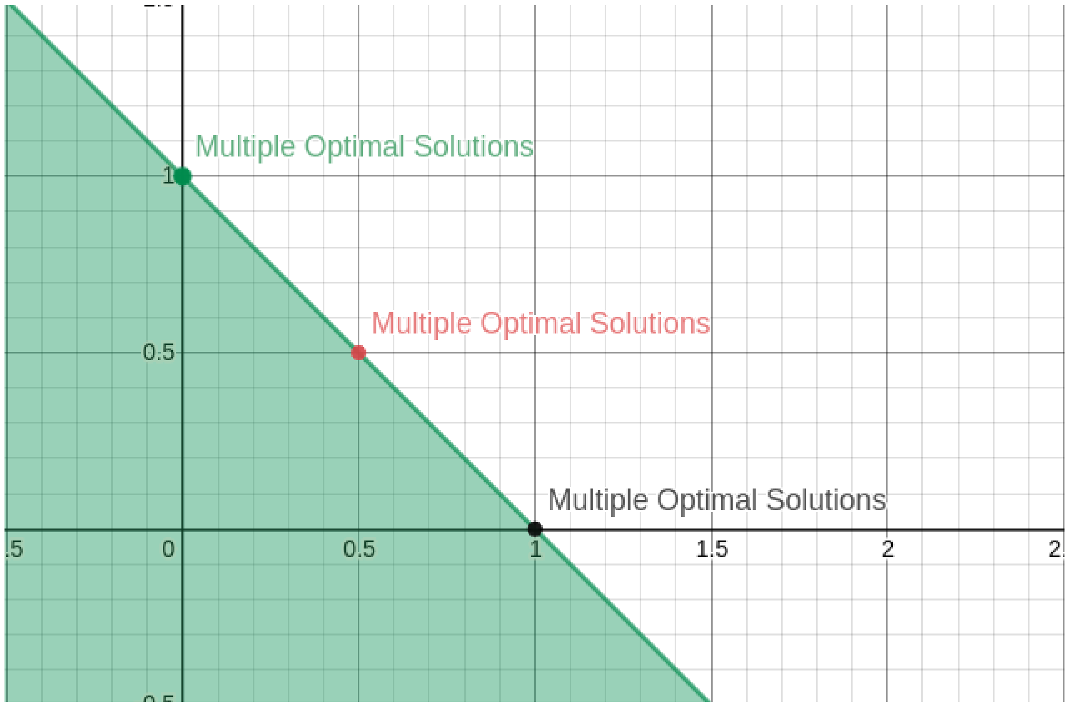

Figure 2: Multiple Solutions Along Segment

- **Cornerpoint Feasible Solution (CPF Solution):** Cornerpoint solutions that border the feasible region.
- Number of cornerpoints $\leq \binom{constraints}{2}$
- Adjacent CPF solutions are connected along the edge of a constraint
- Optimality Test: If a CPF Solution is better than all adjacent CPF solutions, then it must be optimal.
- This works because simplex is always **convex**
- **Convex:** Lines connecting all pairs of nodes fall completely within the enclosed area
- **Boundary** of the feasible region is the parts of the constraint boundaries that are feasible
- CPF solutions in 3 dimensions are the intersection of 3 planes
- In 3 dimensions, each CPF solution has up to 3 possible adjacent solutions
- **Edge:** The feasible portion of the intersection of 2 constraint boundaries
- Number of possible CPF solutions can be exponentially large (num constraints choose num dimensions)
- Not trivial to search through all possible CPF solutions in certain cases
- Bad news: Possible to create an intractable simplex instance
- Good news: In practice, instances are solved very fast
- More Good news: other algorithms can solve LP problems in P!

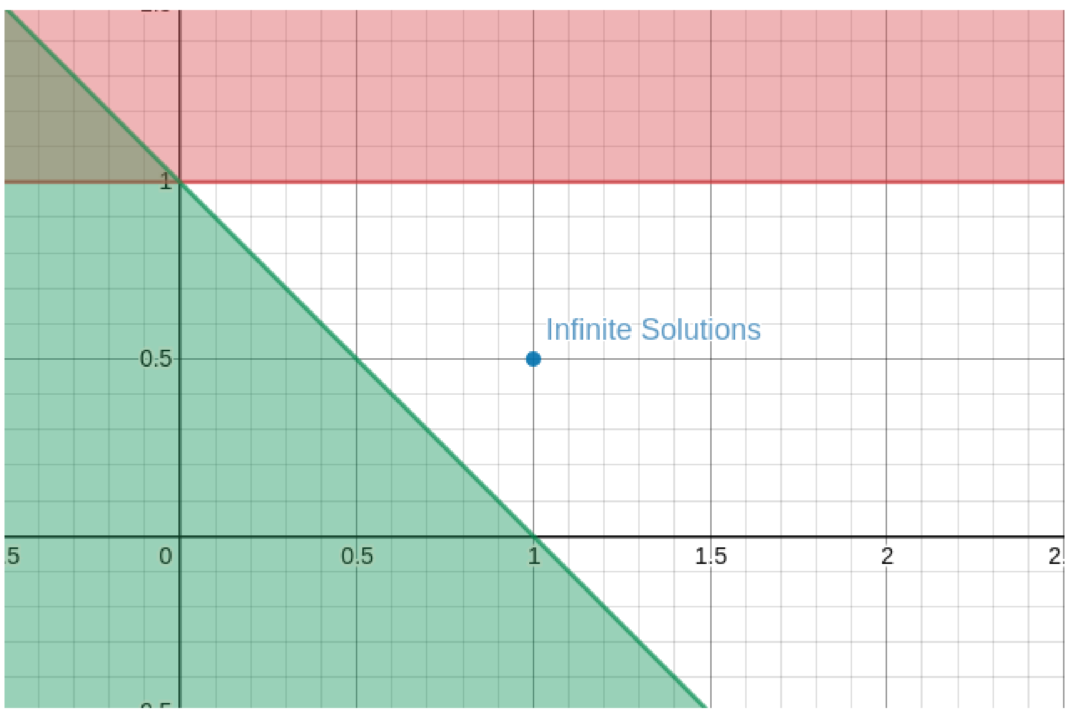

Figure 3: Unbounded Feasible Region

Simplex Algorithm

1. Initialization: pick an initial CPF Solution
2. Optimality test
3. Select new CPF Solution that is adjacent
4. Go back to 2 until optimality test passes.

3 Using LP in Class

AMPL Notes

```
# Write in *.mod file
var x1;
var x2;
maximize objective: x1+x2;
subject to constraint1:
    4*x1-x2 <= 18;
constraint2:
    2*x1+x2 <=  18;
```

1. Write AMPL commands into *.mod file
2. Run `ampl` at unix prompt to launch AMPL

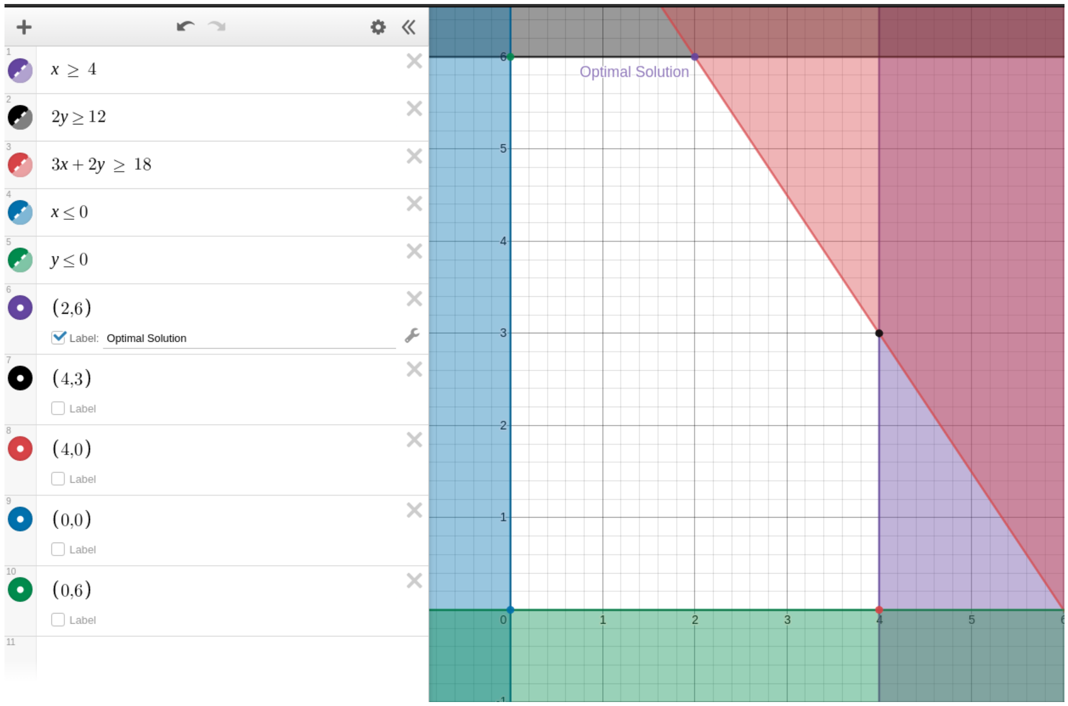

Figure 4: Example LP from Text Book

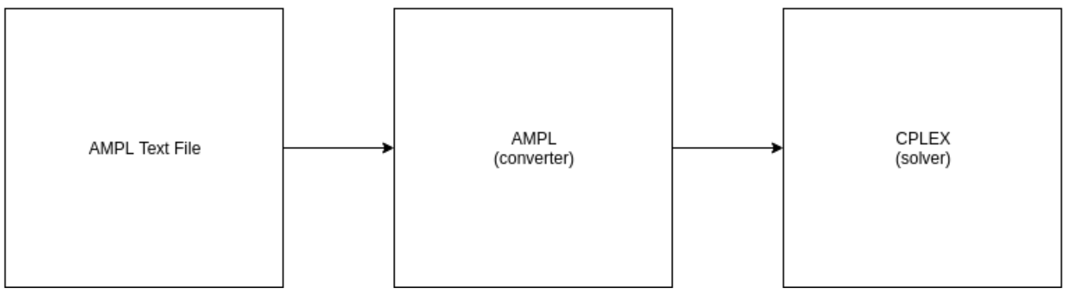

Figure 5: Flow Diagram of Solving Process

3. run `model *.mod` to load model
4. run `data *.dat` to load data
5. run `solve;` to solve
6. Solver spits out # of simplex steps required
7. run `display x1;` to display result
8. run `include *.run` to run script of AMPL commands
9. run `ampl *.run` from unix prompt to run script

AMPL Examples

```
param N=10;
var x {i in 0..N-1};
subject to C1:
    x[0]+3*x[i] <=  10;
c2:
    sum{i in 0..N-1} x[0] = 0;
c3 {j in 0..10}: x[j] = x[j+1];
```

- Variables $\neq$ Parameters
- Parameters are known at solve time - unchanging
- Variables: may change in execution
- Store params in data file
- Store variables in model file