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 15, 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

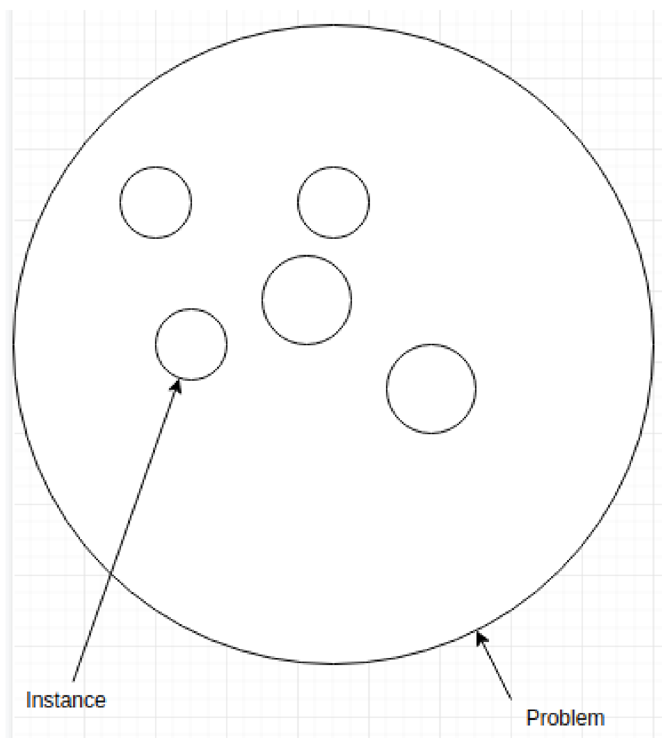

Figure 1: instance_problem

- 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 $f(n) = an^2_b n + c = O(n^2)$. 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

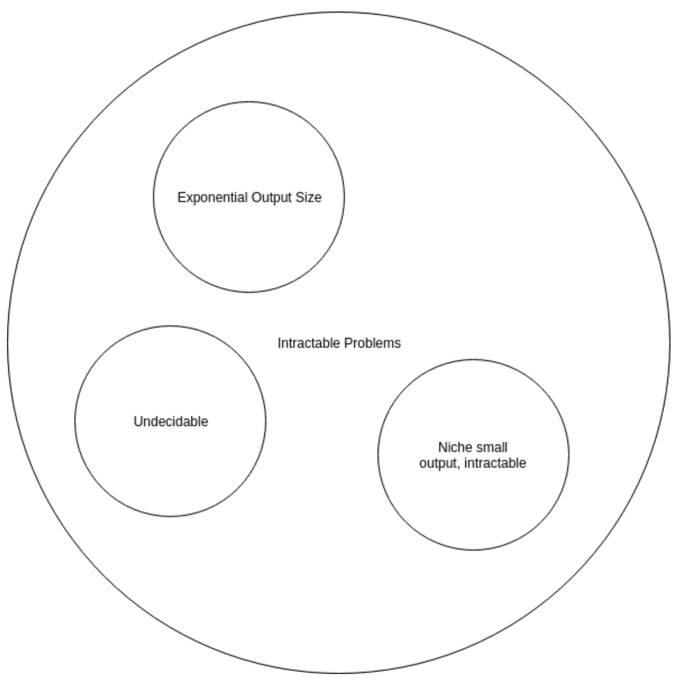

Figure 1: Intractable Problem Categories

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)?

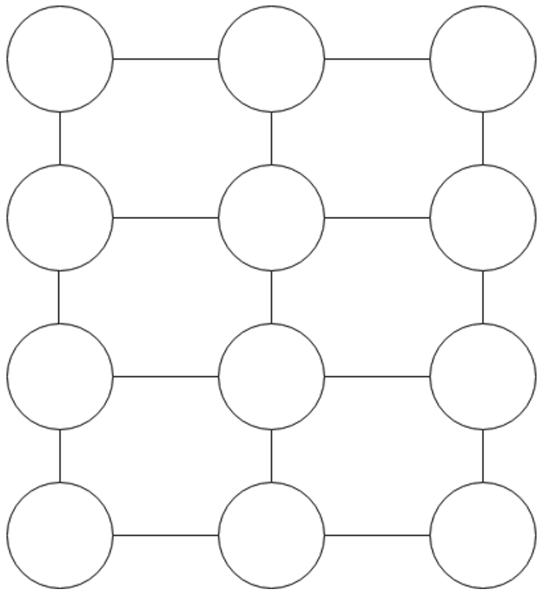

Figure 2: Does this graph have a Hamiltonian Cycle?

- 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

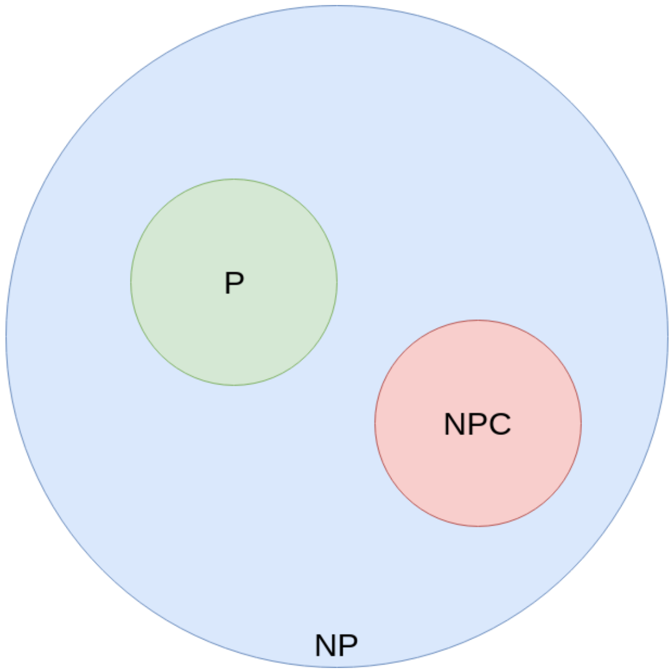

Figure 1: Venn Diagram of P, NP-Complete, and NP

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.

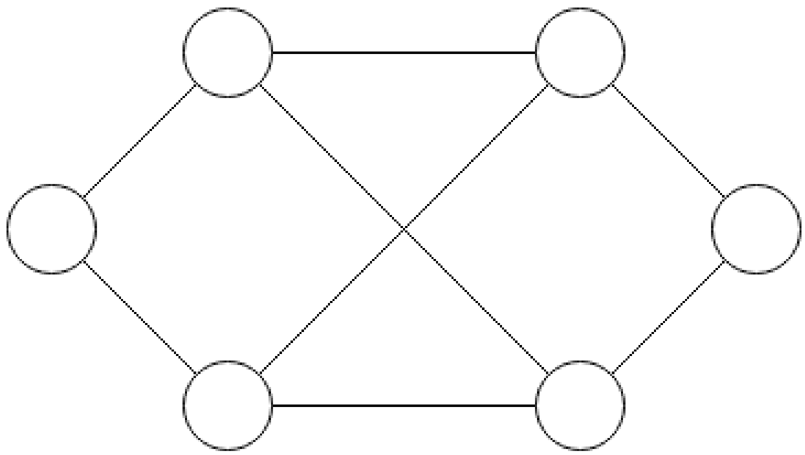

Figure 2: Graph for $HC \leq TSP$ Proof

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:

- $L \in NP$

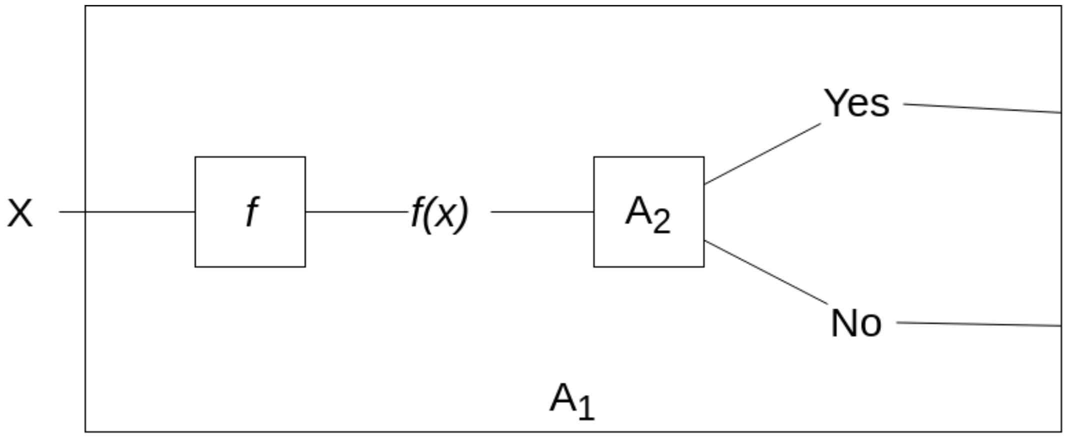

Figure 3: Solving A_1 using A_2 Solver and Reducibility

- 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 13, 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 NP 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

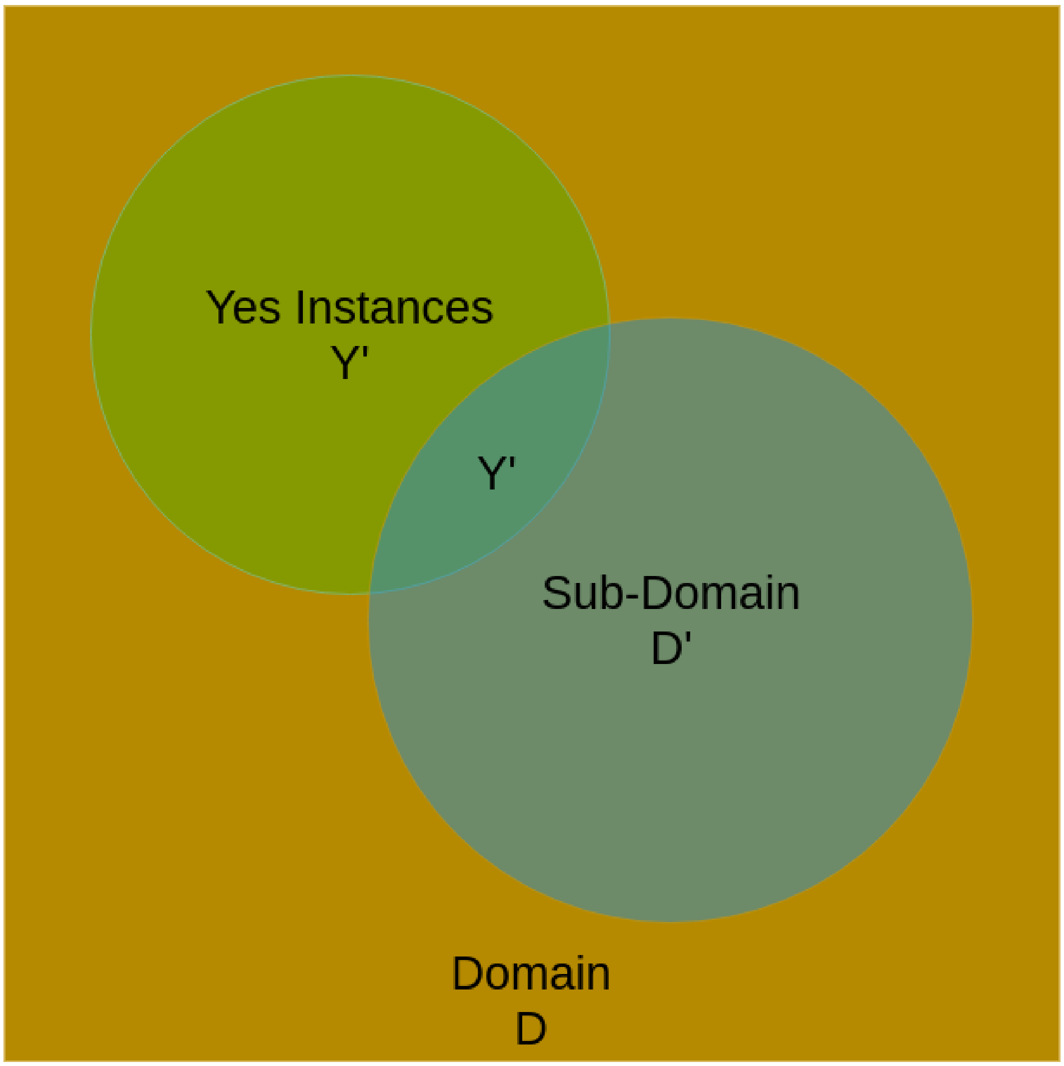

Figure 1: Relation of D, D', Y, Y'

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^{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

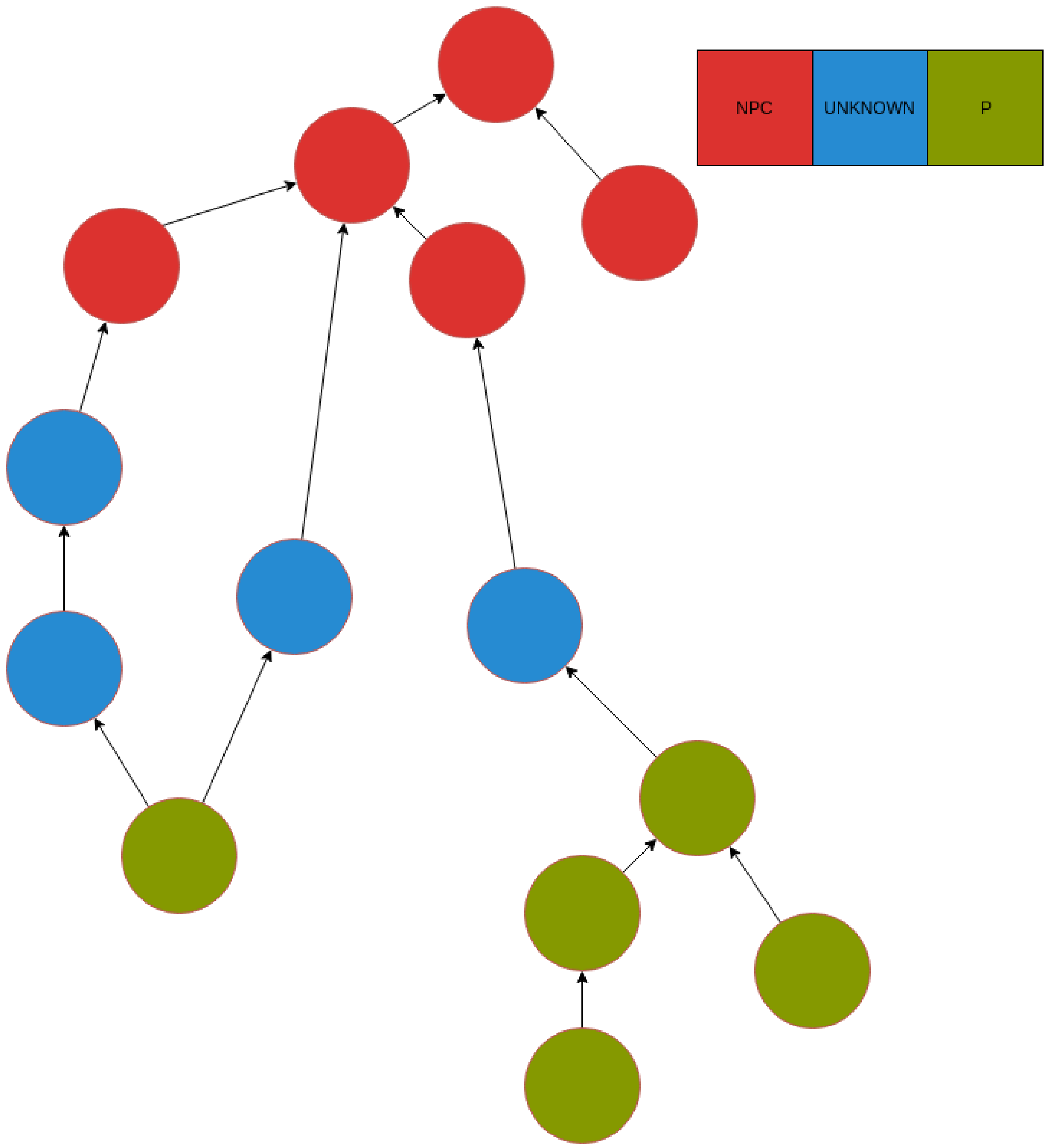

Figure 1: Example Complexity Landscape™

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
```

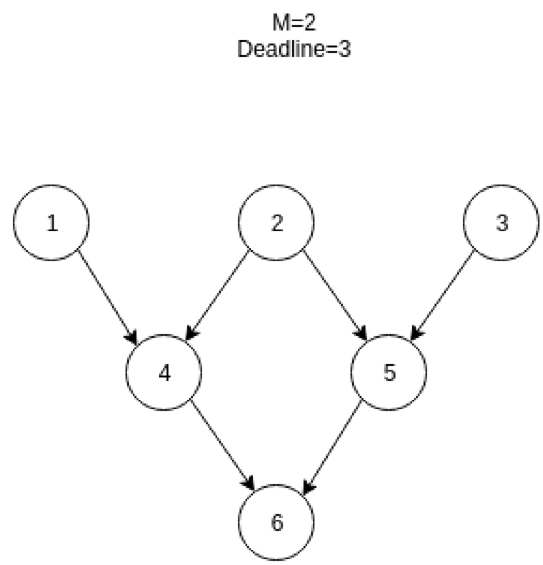

Figure 2: Example PCS Constraint

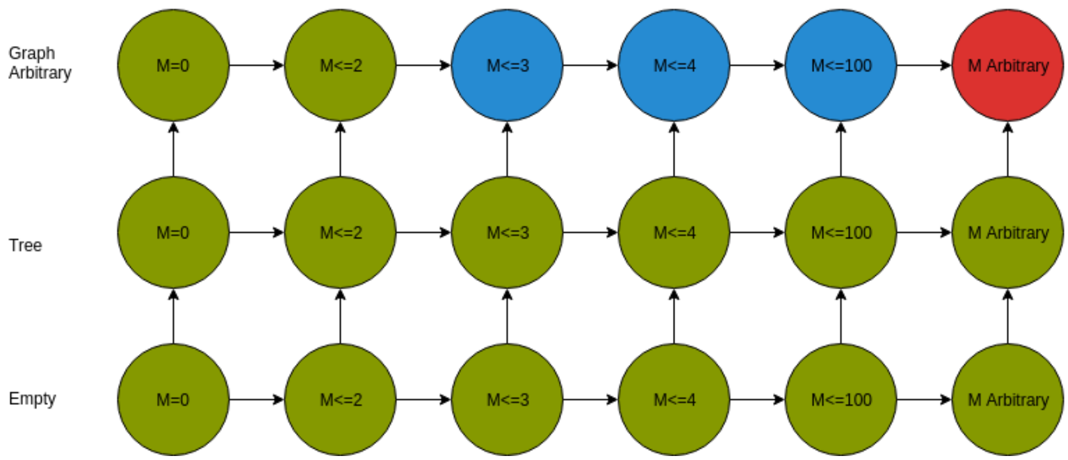

Figure 3: PCS Complexity Landscape™

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

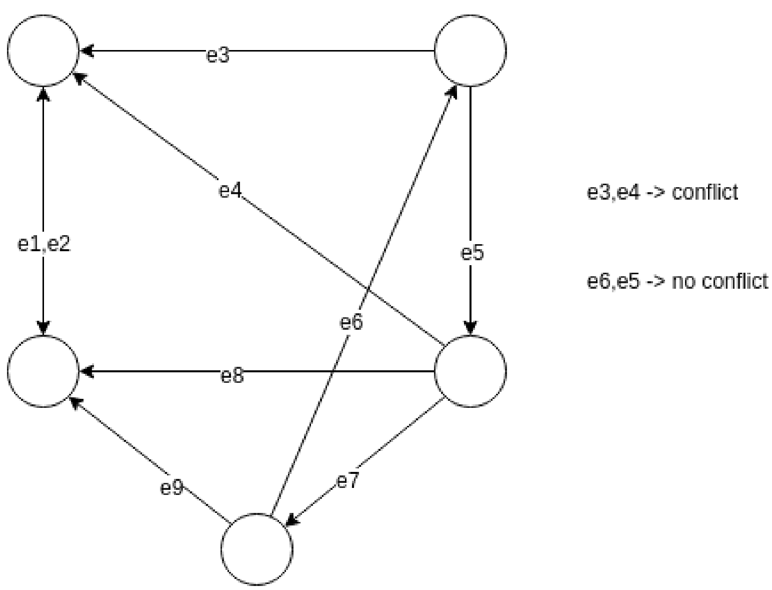

Figure 4: Head Partition Example

```
# 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
```