Backtracking and NP-Complete

Lecture 10

Timothy Kim
GWU CSCI 6212

Backtracking

- Many problems deals with searching for a set of solutions satisfying certain constraints can be solved using Backtracking.
- Following two properties are needed for Backtracking
  - The solution can be expressed as series of choices
  - There exists a function to determine the fitness of the choice. (Fitness Function, Criterion Function, Bounding Function)
- The idea is to map out a decision tree of the problem, and traverse it depth first and at every step call the Fitness Function to see if we are going to right direction.

Generic Algorithm

```
Backtrack(A, i):
  if A is the solution, then return A
  else
    for each x in possible choices:
      if (x fits with A):
        Add x to A
        Backtrack(A, i+1)
```

Sorting via Backtracking

```
Sort(A, S, n):
  if (|A| == n):
    return A
  for each x in S:
    if (fit(A, x)):
      A = A + x
      S = S - x
      Sort(A, S, n)

Fit(A, x):
  return A[last] <= x

Sort([], S, |S|)
```

- Fitness Function:
  - $P : x_i \leq x_{i+1}$

n-Queens Problem

- Place n queens on a n by n chess boards such that no queens can attack any other queens.
- Solution representation:
  - X_i = column # on the ith row
- Example:
  - $X = \{ 2, 4, 1, 3 \}$

	Q		
			Q
Q			
		Q	

n-Queens Fitness Function

- $P : x_i \neq x_j \text{ and } |i - j\,| \neq |x_i - x_j|$

Q			
		Q	
	Q		
			Q

Algorithm

```
nQueens(X, row):
  if row == 4:
    return X
  else
    for each c in {1,2,3,4}:
      if (fit(X, row + 1, c)):
        X[row + 1] = c
        nQueens(X, row + 1)
```

```
fit(X, row, c):
  for i in X.size:
    if Xi == c:
      return false
    if |i - row| == |xi - c|:
      return false
  return true
```

Sum of Subset

- Given a set of weights $W = \{w1, \dots, wn\}$ and a number M, find a subset of W such that the sum of the elements in the subset is equal to M.
- Example:
  - $W = \{11, 13, 24, 7\}$
  - $M = 31$
- Solution as set of choices
  - $X = (1, 1, 0, 1) \mid (0, 0, 1, 1)$

Fitness Function

$$P : \sum_{j=1}^{i} x_j w_j \leq M$$

Algorithm

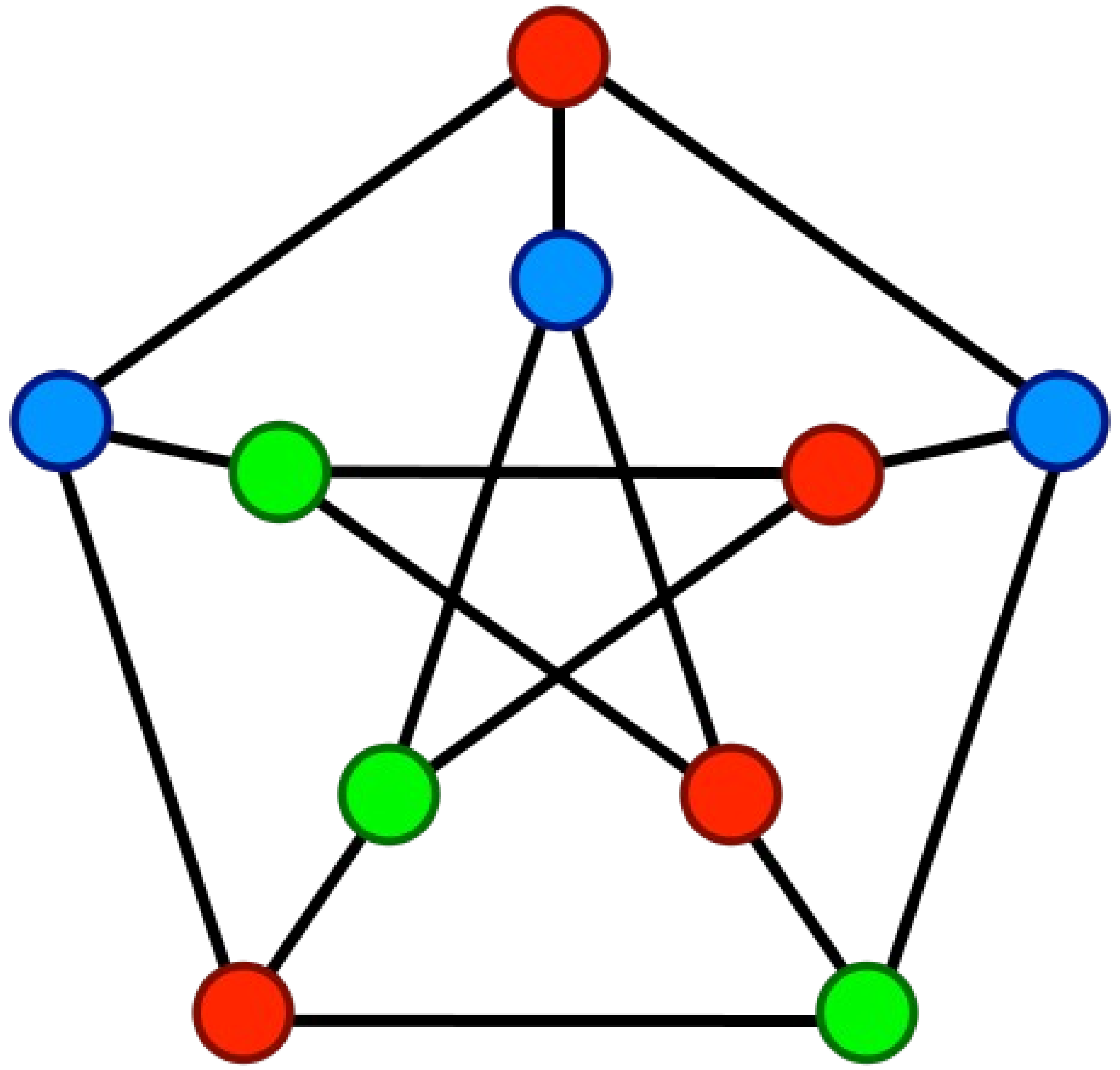
```
Sum(X, W, M):
  X = X + 1
  r = fit(X, W, M)
  if (r == 0):
    return X
  if (r < 0):
    Sum(X, W, M)
  if (r > 0):
    X[last] = 0
    Sum(X, W, M)

Sum([], W, M)
```

```
Fit(X, W, M):
  sum = 0
  for i to X.size:
     sum += X[i] * W[i]
  if sum < M:
    return -1
  if sum == M:
    return 0
  if sum > M:
    return 1
```

Graph Coloring

- Given a graph G and m colors, is there a way of to color the vertices of the graph such that no two adjacent vertices share the same color.
- Solution as choice:
  - C_{i} = color of i^{th} vertex
  - Solution = set of C_i



Fitness Function

- Given a graph that's been colored correctly, how would you check if a next choice of color fits?

Algorithm

- Exercise

Real Brief Introduction to Theory of Computation

- Tractability
- Reducibility
- Completeness
- Computational Class
- NP-Complete

Tractability

- **Definition**: Problem is called tractable if and only if it is a Polynomial-time solvable problem. In other words, there exists an algorithm that correctly solves the problem in $O(n^c)$ time for some constant c.
- n = Input length???
- P = { set of all tractable problems }

Example

- Cycle-free shortest paths in graphs with negative cycle
- Integral Knapsack O(nW)
- Traveling salesman problem?
  - Conjecture: no polynomial-time algorithm exists [Edmonds 65]

Reducibility

- "as hard as"
- Definition: Π reduces to Π' if there exists a subroutine that solves Π' and we can use it to solve Π
- Reduction Time
  - Polynomial-time reduction: if the reduction happens in polynomial time not accounting the time to solve Π'

Example

- Median $\rightarrow$ Selection Algorithm
- Cycle Detection $\rightarrow$ Graph Traversal
- All Pairs Shortest Path $\rightarrow$ Single Source Shortest Path

Completeness

- Suppose Π reduces to Π', if Π is not in P, then neither is Π'
  - "Π' is (at least) as hard as Π "
- Let S = a set of problems
- Π is S–Complete if
  - $\Pi \in$ S and $\forall\ \pi \in S \mid \pi$ reduces to Π in deterministically in polynomial time
  - "Π is the hardest problem in S"

Example

- Let S be all possible problems
- Is Traveling Salesman Problem S-Complete?
  - $\Pi \in$ S is true
  - $\forall\ \pi \in S \mid \pi$ reduces to Π is false
    - There exists a problem that's undecidable (Halting Problem [Turing 36])
    - But TSP is solvable (brute force).

NP

- Π is in NP if
  - size(solution) $= O(n^c)$
  - Solution is verifiable in polynomial time
- TSP
- 3-SAT problem

NP-Complete

- Every problem in NP can be solved by brute-force search, in exponential time.
- Majority of problem are in NP.
- By definition of completeness
  - If there exists an $O(n^c)$ time algorithm that solves one NP-Complete problem, then ALL NP-Complete problem can be solved in $O(n^c)$ time.

NP-Complete

- A single NP-Complete problem encodes simultaneously ALL problems for which a solution can be efficiently recognized.
  - "Universal Problem"
- What?
- Cook 71: Proved that such problem exists
- Karp 72: Proved that 1000s of problem are in NP-Complete (including TSP)

Cook's Theorem

- Every problem in NP can be transformed to the Satisfiability (SAT) problem deterministically in polynomial time.
- Proof: https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem#Proof
- The SAT is the first problem belonging to NP-Complete
- SAT: Given a set U of variables and a collection C of clauses over U. Is there a satisfying truth assignment for C?
- Example:
  - $U = \{x_1, x_2, x_3, x_4\}$
  - $C = \{(x_1 \lor x_2 \lor x_3), (\neg x_1 \lor \neg x_3 \lor \neg x_4), (\neg x_2 \lor \neg x_3 \lor x_4), (\neg x_1 \lor x_2 \lor x_4)\}$
  - Answer $x_1 = T$, $x_2 = F$, $x_3 = F$, $x_4 = T$

Reduction to SAT Example

- **Node Cover Problem**: Given a graph G and an integer K, find a subset of V, S such that (1) for each (u,v) in E, either u or v (or both) is in S and (2) $|S| \leq k$
- Let W be an arbitrary well-formed formula in conjunctive normal form, i.e. sum of products, where W has n variables and m clauses. We then construct G from W as follows.
- The vertex set V(G) is defined as $V(G) = X \cup Y$, where $X = \{x_i, \neg x_i \mid 1 \leq i \leq n\}$ and $Y = \{p_j, q_j, r_j, \mid 1 \leq i \leq m\}$. The edge set of G is defined to be $E(G) = E_1 \cup E_2 \cup E_3$, where $E_1=\{(x_i, \neg x_i) \mid 1 \leq i \leq n\}$ and $E_2 = \{(p_j, q_j), (q_j, r_j), (r_j, p_j) \mid 1 \leq i \leq m\}$ and E_3 is defined to be a set of edges such that p_j, q_j, r_j are respectively connected to $c1_j$, $c2_j$, and $c3_j$ where $c1_j$, $c2_j$, and $c3_j$ denote the first, second and the third literals in clause C_j.

Reduction to SAT Example

For example, let $W = (x_1 + x_2 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)$. Then G is defined such that $V(G) = \{x_1, \overline{x}_1, x_2, \overline{x}_2, x_3, \overline{x}_3, p_1, q_1, r_1, p_2, q_2, r_2, p_3, q_3, r_3\}$ and $E(G) = \{(x_1, \overline{x}_1)$, $(x_2, \overline{x}_2)$, $(x_3, \overline{x}_3)$, (p_1, q_1), (q_1, r_1), (r_1, q_1), (p_2, q_2), (q_2, r_2), (r_2, p_2), (p_3, q_3), (q_3, r_3), (r_3, p_3), (p_1, x_1), (q_1, x_2), (r_1, x_3), $(p_2, \overline{x}_1)$, (q_2, x_2), $(r_3, \overline{x}_3)$, $(p_3, \overline{x}_1)$, $(q_3, \overline{x}_2)$, $(r_3, \overline{x}_3)\}$.

Reduction to SAT Example

We now claim that there exists a truth assignment to make $W = T$ if and only if G has a node cover of size $k = n + 2m$.

To prove this claim, suppose there exists a truth assignment. We then construct a node cover S such that $x_i \in S$ if $x_i = T$ and $\overline{x}_i \in S$ if $x_i = F$. Since at least one literal in each clause C_j must be true, we include the other two nodes in each triangle (i.e., p_j, q_j, r_j) in S. Conversely, assume that there exists a node cover of size $n + 2m$. We then note that exactly one of $x_i, \overline{x}_i$ for each $1 \leq i \leq n$ must be in S, and exactly two nodes in p_j, q_j, r_j for each $1 \leq j \leq m$ must be in S. It is then easy to see the S must be such that at least one node in each p_j, q_j, r_j for $1 \leq j \leq m$ must be connected to a node x_i or $\overline{x}_i$ for $1 \leq i \leq n$. Hence we can find a truth assignment to W by assigning x_i true if $x_i \in S$ and false $\overline{x}_i \in S$.

P = NP

- P = set of all problems that are SOLVABLE in polynomial time
- NP = set of all problems that are VERIFIABLE in polynomial time
- The big question: P = NP ?
  - Find a single reduction, $\Pi \rightarrow \Pi'$ where Π in in NP-Complete

- NP = "non-deterministic polynomial"

Final Exam Topics

- Greedy Algorithm
  - Dijkstra's
- Dynamic Programming
  - General Idea
  - Maximum Weight Independent Set
  - Integral Binary Knapsack
  - Optimal Binary Search Tree
  - Bellman-Ford
  - Floyd-Warshall
  - Johnson's
- Backtracking
  - General Idea
  - n-Queens
  - Sum of subsets
  - Graph Coloring
- Computational Theory
  - General Idea
  - Definitions