Introduction to Approximation Algorithms

Ron Parr
CPS 570

Covered Today

• Approximation in general
• Set cover
• A greedy algorithm for set cover
• Submodularity
• Generic, greedy algorithm exploiting submodularity
Some Intractable Combinatorial Optimization Problems

• Find the lowest cost traveling salesman tour

• Color a graph with the fewest possible colors

• Cover with the lowest number of vertices/sets

Set Cover

• Input:
  – A set of atoms: \( A=a_1...a_n \)
  – A set of sets: \( S=s_1...s_m \)
  – Each set contains 1 or more atoms

• Optimization question: Can you choose \( k \) elements from \( S \) such that every element of \( A \) is in at least one of these \( k \)? (This is a called a cover.)

• Decision question: Exist a cover of size \( k \) or less?

• NP-hard
Set Cover Example

14 atoms
5 sets

Hardness of Set Cover

- Karp showed that set cover is NP-complete (classic paper on reading list)
- Satisfiability reduces to clique
- Clique reduces to node (vertex) cover
  - Node cover reduces to set cover
**Node (vertex) Cover**

- **Input:**
  - Graph $G = E, V$

- **Optimization question:** What is the smallest set of vertices such that every edge is incident upon one of the vertices

- **Decision question:** Does there exist a set of vertices of size $k$ such that every edge is incident on at least one vertex in $k$

---

**Reduce Node Cover to Set Cover**

- **Remember:** Must solve node cover w/set cover
- **For each edge in the node cover problem, we create an atom in the set cover problem**
- **For each node in the node cover problem, we create a set s.t. elements of the set correspond to edges incident to the node**

- **Observe that a set cover of size $k$ exists iff a node cover of size $k$ exists**
Reduction Example

- Edges in node cover -> atoms (a1...a3) in set cover
- Each node in node cover -> set with all adjacent edges in set cover (c1..c3)

Real Problems Abstracted by Set Cover

- Sensor placement:
  - You have sensors to place in m different locations
  - Each location can observe some fraction of your n targets
  - Find the most efficient sensor allocation to see all targets

- Buying bundles of goods
  - Different vendors offer package deals on different combinations of products (flat rate shipping)
  - Buy all the products you need in the smallest number of transactions

- Choosing advertising outlets
  - Different stations/newspapers/sites cover different, possibly overlapping markets
  - Try to cover markets with smallest number of ads

- Choosing test cases for code/hardware/exams
  - Different tests exercise different (overlapping) parts of the system
  - Try to verify system in smallest number of tests
Why use approximation?

• Many problems we want to solve are NP-hard optimization problems w/associated NP-complete decision problems

• Different notions of approximation
  – Search for a “pretty good” answer
  – Return an optimal answer in some cases (fail in others?)
  – Return an answer that is an additive factor from optimal:
    \[ \text{result} = \text{optimal} +/\ - \varepsilon \]
  – Return an answer that a multiplicative factor from optimal:
    \[ \frac{\text{result}}{\text{approximation}} = \varepsilon \]
  – For a given resource level, achieve a lower performance value?
  – For a given performance level, consume more resources?

So, what do we do in, e.g., set cover?

• Settle for a larger k (more sets)?
  – What if we don’t need the absolute smallest k?
  – Is there an algorithm that gives something close to the smallest but still covers everything?

• Settle for less than full coverage
  – What if we have only k resources?
  – Is there an algorithm that gives us something close to the best coverage using only k?
Greedy Algorithms

• Greedy algorithms are a general class of algorithms that, *loosely speaking*, make a choice that gives *maximal short term improvement*, *without considering subsequent choices*.

• Examples of greedy behavior:
  – Picking the class that is most interesting to you first (ignoring that this might cause scheduling problems with other classes)
  – Positioning a sensor so that it sees the highest number of targets (while ignoring subsequent choices)

Greedy Set Cover

• Repeat until done*
  – For each set not added, check how many previously uncovered atoms it would add
  – Add the set with the *biggest increase in the number of atoms covered*

• *What is “done”*
  – Max of k elements added, *or*
  – All elements covered
What does greedy do here?

What price greed?

• Assume we have a budget of $k$

• Optimal picks: $O_1...O_k$, covering $n$ atoms

• Greedy picks $G_1...G_k$, covering $x$ atoms

• What is the relationship between $x$ and $n$?
What price greed (2)?

- $o_i =$ number of new elements covered by $O_i$
- $g_i =$ number of new elements covered by $G_i$
- “new” means not previously covered by $1...i-1$

- $n = o_1 + o_2 + ... + o_k$
- $x = g_1 + g_2 + ... + g_k$

What price greed (3)?

- Suppose $o_i > g_i$, $i > 1$
- Q: Why didn’t greedy pick $o_i$?
- A: The only reason would be if greedy already covered $o_i - g_i$ of the elements in $o_i$ in some $g_j$, $j < i$
- $x \geq (o_1 - g_1) + (o_2 - g_2) + ... + (o_k - g_k) = n - x$
- $2x \geq n$
- $x \geq n/2$

- Conclusion: For fixed $k$, greedy gets a least half as much coverage as optimal
What about minimizing \( k \) to achieve full coverage?

- Suppose optimal coverage uses \( k \) sets to cover \( n \) atoms
- We run greedy until it covers everything, taking \( h \geq k \) sets

- Analyze greedy's \( h \) choices in batches of \( k \)
  - Greedy covers at least \( n/2 \) in first batch of \( k \)
  - Second batch of \( k \) covers at least half of remaining atoms. Why?
  - Same analysis can be repeated.

- Conclusion: greedy requires at \( O(k \log n) \) sets

- Note: Our bounds are **not tight** in this case. Better proof exploiting submodularity is possible.

Applying to Other Problems

- If we have a good approximation scheme for one NP-hard problem, does this imply a good approximation scheme for others? (e.g. transform to set cover, then approximate the transformed problem)

- Depends upon what you mean by “good”...

- The polynomial factor can be a killer here

- Conclusion: Approximation algorithms will tend to be problem specific unless one discovers a more general approach to approximation
Submodularity

• f is a function defined on sets
• Monotone if:

\[ X \subseteq Y : f(Y) \geq f(X) \]

• Submodular if

\[ X, Y \subseteq \Omega, X \subseteq Y, z \notin Y : f(X \cup \{z\}) - f(X) \geq f(Y \cup \{z\}) - f(Y) \]

In English

• Monotonicity: Bigger is better
  (though not strictly)

• Submodularity:
  – Adding to a subset has at least as much “bang” as adding to a superset, or
  – Diminishing returns for adding to bigger sets
Set Cover?

- Does set cover fit this framework?
- $f =$ number of atoms covered

- Is it monotone?
- Is it submodular?

Maximizing Monotone Submodular Set Functions

- Goal: Given budget of $k$ sets, maximize $f$
- This is NP-hard in general 😞

- Greedy algorithm for maximizing $f$ that is a
  - Non-negative
  - Monotone
  - Submodular
  set function is a $1 - 1/e (=0.63)$ factor from optimal for budget $k$
- Similar argument to set cover to gives a resource bound
- Proof in reading, similar to our 2X bound, but a little more subtle

- Provides a generic procedure for analyzing greedy algorithms for certain classes of hard problems 😊
Greedy Submodular Maximization

• Input: set of sets $\Omega$, score function $f$
• $X = \{\}$
• Repeat until “done”
  – Find set $\omega$ in $\Omega$ that maximizes $f(X+\omega)$
  – Add $\omega$ to $X$

• “done”:
  – $|X| = k$, or
  – $f(X) =$ some target value

Greedy Set Cover and Submodularity

• Our greedy algorithm for set cover can be understood as an instance of the greedy approach for submodular set functions

• Conclusion: We get a tighter bound for free!
• $(1-1/e > \frac{1}{2})$
Exploiting Submodularity

• Frequently used to justify greedy approaches that otherwise would have had computation/implementation ease as their only justification

• Impactful in, e.g., sensor network community

Conclusions

• Avoid worst consequences NP-hardness with clever approximation algorithms (or clever analysis of simple algorithms)

• Caveats:
  – Not all problems admit good approximate solutions
  – Specific approximation techniques for one problem don’t necessarily apply to others

• Some generic approaches exist:
  – Greedy algorithms sometimes do well
  – Submodularity provides a generic framework for analyzing certain types of greedy algorithms
  – Other families of approaches exist as well – rounding, LP relaxations, etc.