Introduction to Approximation Algorithms

Ron Parr
CPS 570

Covered Today

• Approximation in general
• Set cover
• A greedy algorithm for set cover
• Submodularity
• A generic, greedy algorithm exploiting submodularity

Why use approximation?

• Lots of problems we want to solve are NP-hard optimization problems, often with 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: result = optimal +/- ε
  — Return an answer that a multiplicative factor from optimal: result/approximation = ε
  — For a given resource level, achieve a lower performance value?
  — For a given performance level, consume more resources?

Set Cover

• Input:
  — A set of atoms: \( S = s_1 ... s_n \)
  — A set of sets: \( C = c_1 ... c_m \)
  — Each set contains 1 or more atoms

• Optimization question: Can you choose \( k \) elements from \( C \) such that every element of \( S \) is in at least one of these \( C \)? (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

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 (or newspapers) cover different, possibly overlapping markets
  – Try to cover markets with smallest number of ads

So, what do we do?

• Settle for a larger k?
  – What if we don’t need the absolute smallest k?
  – Is there an algorithm that gives something close to the smallest?

• 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 we can hope for using 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 \( G_i \)

• \( n = o_1 + o_2 + ... + o_k \)
• \( x = g_1 + g_2 + ... + g_k \)
What price greed (3)?

- Suppose $o_i > g_i$
- 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 $Q_i$ in some $G_j$, $j < i$
- $x \geq (o_1 - g_1) + (o_2 - g_2) + \ldots + (o_k - g_k) = n - x$
- $2x \geq n$
- $x \geq n/2$

- Conclusion: For fixed $k$, greedy gets at least half as much coverage as optimal

What about minimizing $k$?

- Suppose optimal coverage uses $k$ to cover $n$ atoms
- Assume we run greedy until it covers everything, taking $h > k$ resources
- 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 most $k \times \log_2 n$ resources

- Note: Our bounds here are not tight. Better proof exploiting submodularity is possible.

Applying to Other Problems

- If we have a good approximation scheme for one NP-complete problem, does this imply a good approximation scheme for others?
- 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
- Submodular if:
  \[ X, Y \subseteq \Omega, X \subseteq Y : f(X \cup \{z\}) - f(X) \leq f(Y \cup \{z\}) - f(Y) \]
- Monotone if
  \[ X \subseteq Y : f(Y) \geq f(X) \]
Submodularity in English

• Adding to a subset has more “bang” than adding to a superset, or
• Diminishing returns for adding to bigger sets
• Monotonicity in English: Bigger is better (though not strictly)

Set Cover?

• Does set cover fit this framework?
• \( f = \text{number of atoms covered} \)
• Set \( \Omega = C \)
• Is it submodular?
• Is it monotone?

Maximizing Monotone Submodular Set Functions

• This is NP-hard in general 😞
• Greedy algorithm for maximizing monotone submodular set functions is a 1-1/e factor from optimal
• Can use similar argument to set cover to get a resource bound
• Proof in reading, similar to our 2X bound, but a little more subtle
• This provides a generic procedure for analyzing greedy algorithms for certain classes of hard problems 😊

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!
• \((1-1/e > \frac{1}{2})\)
Conclusions

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

- Caveats:
  - Not all problems admit good approximate solutions
  - Approximation techniques for particular problems don’t always carry over 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.