Presentation Script

About

The content of this presentation mainly covers Chapter one, and a little bit of Chapter two of the book The Design of Approximation Algorithms. So there may be sentences in this script that are paraphrased from the content of the book. The presentation is only given out in a private group meeting to enlighten students with some notions on the area of approximation algorithms.

The link to the book is listed as follow: http://www.designofapproxalgs.com/

Context

This presentation is going to cover about some really basic stuffs related to approximation algorithms. The slide will start off by briefly telling you why we need approximation, next we'll introduce the set cover problem and tackle this problem with two distinct techniques, linear programming and the greedy algorithm. For the final part, we will illustrate the local search method and use the epic travel salesman problem to show you how it works.

1. Page1 Introduction

So why approximation? One of the main reasons is fairly simple. Most of the interesting optimization problems are NP-hard, for these kinds of problems there's an old engineering slogans saying "Fast, Cheap, Reliable" Choose two. In other words, we just can't have algorithms that can find you the optimal solutions, in polynomial time and for any instance. At least one of these requirements has to be relaxed.

Therefore, when dealing with many real-world problems, one of the most common approach is to relax the requirements of finding an optimal solutions, and instead settle for a solution that is "good enough". In the following slides we will define these "relaxed" algorithms as an alpha-approximation algorithms.

An alpha-algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem, it produces a solution whose value is within a factor level of alpha for the problem. Where for minimization problems, alpha will be larger than 1 and for maximization problems it will be less than 1.

2. Page2 Set Cover Problem

The first sample problem is the weighted set cover. In this problem, we're given a universe that contains m elements and n subsets of those elements where each subsets have a corresponding cost. You must select a number of these sets so that the sets you have picked contain all the elements that are contained in the universe, and minimize the total cost of the chosen subset.

Now why is solving this problem useful? Consider the following scenario. As a procurement manager of a company, you need to buy a certain amount of varied

supplies and there are suppliers that offer various deals for different combinations of materials. In this case you could use set covering to find the best way to get all the materials while minimizing the total cost.

3. **Page3 Linear Model**

One way to deal with it this problem is to formulate a linear model. In the case of the set cover problem, we need to decide which subsets S to use in the solution. We create a decision variable x to represent this choice, that is, x_j is to be 1 when S_j is chosen and 0 otherwise.

We also want to make sure that any feasible solution corresponds to a set cover. In order to do so, we have to add additional constraints to ensure that every element in the universe is covered. It must be the case that at least one the subsets S containing the universe's element i is selected. This part may be a bit confusing, so an example is given at page 4.

4. **Page4**

Example for the constraint part of the linear model (scripts omitted).

5. **Page5 Linear Programming**

But, In general, integer programming cannot be solved in polynomial time, however, linear programming are polynomial time solvable.

6. **Page6**

Therefore using a polynomial time solvable relaxation of a problem in order to obtain a lower bound for minimum problem or an upper bound for maximum problem is often extremely useful. And it can also be used to derive approximation algorithms for the set cover problem, which we'll discuss in the next slide.

In this case, the linear model is relaxed from an integer programming to a linear programming problem by converting the constraints from x has to be 0 or 1 to only x has to be larger than 0.

7. **Page7 Deterministic Rounding**

Suppose that we solve the linear programming relaxation of the set cover problem. Let x^* denote an optimal solution to the LP (Linear Programming). How then can we recover a solution to the set cover problem? A very easy way to do it is by deterministic rounding. Given the LP solution, we include subset S_j in our solution if and only if x_j^* is larger than 1 over f, where f is the maximum number of subsets in which any element appears.

8. **Page8 Deterministic Rounding Theorem**

Theorem (scripts omitted).

9. **Page9**

Proof (scripts omitted).

10. **Page10 Greedy Algorithm**

The next approximation technique is the greedy algorithm.

By definition in plain English, it is a process that looks for an easy to implement solution to complex, muti-step problems by deciding which next step will provide the most obvious benefit. The algorithm doesn't consider the problem as a whole, and once a decision at the certain step has been made, it is never reconsidered.

While this method can be straightforward and easy to understand, the disadvantage is that the most optimal short term solutions may lead to the worst long term outcome.

11. Page11

Let's consider the same set cover problem, only this time without the weight/cost that were assigned to each subset. Therefore the objective of this simplified version of the set cover problem has changed to: You must select a minimum number of these sets so that the sets you have picked contain all the elements that are contained in the universe.

12. Page12

Example of unweighted set cover problem and code demo using greedy algorithm (scripts omitted).

13. Page13 Greedy Algorithm Theorem

Theorem (scripts omitted).

14. Page14

Let the universe U contain n points, and suppose that the optimal solution has size m. The first set picked by the greedy algorithm has size at least n/m. Therefore, the number of elements of U we still have to cover after the first set is picked is n-n/m.

We are then left with n1 elements to cover. At least one of the remaining sets Si must contain at least n1/(m−1) of such points them because otherwise the optimum solution would have to contain more than m sets. After our greedy algorithm picks the set that contains the largest number of uncovered points, it is left with n2 ≤ n1 − n1/(m − 1) uncovered elements.

In general we'll have the following inequality for each iteration of the algorithm.

So suppose that it takes k stages to cover the universe U, this will lead us to $n_k \leq n(1-1/m)^k$, and we need this to be less than one, since we must have at least less than one elements to cover after the full thing is finished.

Other Reference for this section:

http://pages.cs.wisc.edu/~shuchi/courses/787-F07/scribe-notes/lecture02.pdf

15. Page15

Proof (scripts omitted).

16. Page16 Local Search

The final technique that we'll cover is the local search method.

The simplest way for presenting the intuition behind Local Search is the following: imagine a climber who is ascending a mountain on a foggy day. She can view the slope of the terrain close to her, but she cannot see where the top of the mountain is. Hence, her decisions about the way to go must rely only upon the slope information. The climber has to choose a strategy to cope with this situation, and a reasonable idea is, for example, choosing to go uphill at every step until she reaches a peak. However, because of the fog, she will never be sure whether the peak she has reached is the real summit of the mountain, or just a mid-level crest.

Other Reference for this section:

http://www.diegm.uniud.it/digaspero/papers/DiGasperoPhDThesis.pdf

17. Page17 Random Hill Climbing

There're many different types of algorithms under the Local Search family, for that we will only introduce the most basic one, Random Hill Climbing. This algorithm relies on the basic idea chosen by the climber in the metaphor proposed above: First it is given a random initial solution, then at each step of the search, a move that "leads uphill" is performed. This procedure will continue until the user-defined criteria is met, in this case it is the iteration of the search. As to how to choose the next step for the search, we will use the travel salesman problem to demonstrate.

18. Page18 Travel Salesman Problem

In the traveling salesman problem, there is a given set of n cities, and the input consists of a symmetric n by n matrix C = (c_{ij}) that specifies the cost of traveling from city i to city j. By convention, we assume that the cost of traveling from any city to itself is equal to 0, and costs are nonnegative; the fact that the matrix is symmetric means that the cost of traveling from city i to city j is equal to the cost of traveling from j to i.

Our objective for the question is then what is the shortest possible route that visits each city exactly once and returns to the origin city?

Code and demo.