Combinatorial optimization

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 16

Combinatorial optimization problem statement

We still want to optimize (minimize) our multi dimensional merit function E

Find $\vec{x}$ that minimize $E(\vec{x})$ subject to $g(\vec{x}) = 0, h(\vec{x}) \leq 0$

The only difference values of $\vec{x}$ are discrete. I.e. any component of $\vec{x}$ is not continues and can take a **countable** set of different values.

In this case we cannot run our golden search algorithm or anything else which assumes continuous space for $\vec{x}$.

Instead we have to find a method to search through discrete sets of all possible input values.

Example: Backpack problem

Suppose you have a backpack which has a given size (volume). You have a set of objects with given volumes and values (for example their cost).

Our job is to find a such subset of items that still fits in the backpack and has the maximum combined value.

In simple case we will assume that every items happen only once. Then our job is to minimize

$$E(\vec{x}) = \sum value_i x_i$$

Subject of the following constrains

$$\sum volume_i x_i \leq \mathrm{BackpackSize}$$

Where $x_i = (0 \text{ or } 1)$ i.e. do we take this object or not

Brute force optimization

With this approach we will just try all possible combinations of items and find the best of them.

Notice that if there is N objects we have 2^N of all possible combinations to choose from.

So the size of the problem space and thus probing time grows very fast.

Brute force optimization continued

First we need an algorithm which generates all possible permutations of the items.
The below method goes back to 14th century India. It generates permutations in the lexicographical order.

1. find the largest index k such that $p(k) < p(k+1)$.
   - if no such index exists, the permutation is the last permutation.
2. find the largest index l such that $p(k) < p(l)$.
   - There is at least one $l = k + 1$
3. swap $a(k)$ with $a(l)$.
4. reverse the sequence from $a(k+1)$ up to and including the final element $a(end)$.

See the complimentary code 'permutation.m'

Brute force optimization continued

Then the main algorithm for checking and going through permutations see 'backpack.m' listing.
It is deliberately super inefficient with worse case time execution
$\sim N \times N! \gg 2^N$
Sample run

```
backpack_size=7;
volumes=[  2,  5,  1,  3, 3];
values =[ 10, 12, 23, 45, 4];
[pbest, max_fitted_value] = backpack( backpack_size, v

pbest = [1 3 4]
max_fitted_value = 78
```