Combinatorial optimization

Francesco Tomaselli

October 4, 2021

Contents

1 Graphs

1.1 Ways to generate a random graph

What we want to do is to generate a random graph with a given density. Formally, we want a $G(V, A)$ with density:

$$\rho = \frac{r}{n(n-1)}$$

Thus the number of required edges are

$$p = \rho m$$

Edges coding We must image a coding between natural numbers and edges, as we need it for the following algorithms, some examples are:

- ...

First algorithm The first idea is to select at random r edges from the set of all possible m ones.

The time complexity is $O(r)$ and the space complexity is $O(r)$.

Second algorithm A second algorithm could be selecting r edges at random, maintaining a set or vector with boolean values to avoid repetitions.

The space complexity is $O(m)$ with a boolean vector, or $O(r)$ with a set. We could use a bitvector to reduce memory, while the time complexity requires a more detailed analysis.

Let W_k be the random variable that indicates how many conflicts we get selecting at random from m with k already selected edges.

...

$E[W] = p$

Time complexity now now depends on the structure we use to store the already seen edges. The worst case complexity is infinite.

Third algorithm We can allocate an array with m elements, select a random i and swap it with the first element. Proceed with selecting a random number between in $[2, m]$ and swap it with the second element. So on until we have p edges.

The time complexity is $O(p)$ and so is the space complexity is $O(m)$.

Fourth algorithm An improvement can be made in the space of the third algorithm, we can store at most $O(p)$ elements taking care of the swaps. An extraction then search in the p auxiliary elements, and if found it uses the swap indicated there, if not, it uses the extracted random number as edge.

Time complexity now depends on the way we store the additional p elements. If we store them in a linked list, time complexity becomes $O(p^2)$ and space $O(p)$.