CS4618 Artificial Intelligence I
Term 2012/2013, 1st Period
http://www.cs.ucc.ie/∼tj2/cs4618/

Thomas Jansen
University College Cork
October 24th, 2012

Assignment 5

Topic 'Application of Tail Bounds'

Problem 1 We want to make a promise about a specific randomised search heuristic (RSH) on a specific problem. We want to prove a statement 'This RSH finds an optimal solution for this problem in at most t steps with probability at least p'. Clearly, we would like to have such a statement for small values of t and values of p that are close to 1. But there is a trade-off: If we decrease t we have to accept smaller values of p.

Assume we know the following about this RSH on this problem. The expected number of steps the RSH takes to solve this problem is at most 5000. This holds even for all possible choice of the initial search points.

Prove

$$\text{Prob}\,(\text{RSH optimises problem in at most } t \text{ steps}) \geq p$$

for appropriate values of t and p. If you can, prove a general statement where the value of p is a function depending on t. Remember: The closer p is to 1 the better.

Topic 'Random Processes'

Problem 2 Remember MAX-SAT (compare with Problem 1 on Assignment 4). Now assume that all clauses contain exactly two variables and assume that an assignment that satisfies all clauses at the same time exists. Consider the following algorithm. It starts with a random assignment. If the assignment satisfies all clauses the algorithm stops. Otherwise it goes through the clauses and considers the first one that is not satisfied. This clause contains two variables. The algorithm picks one of them uniformly at random and changes its value. Now it continues.

Give an upper bound on the number of assignments the algorithm considers before stopping. Hint: Try to find a random process from the lecture that matches the behaviour of this algorithm.

Problem 3 **Implementation Task**

Standard bit mutations flip each bit in a bit string of length n independently with a fixed probability p. Usually, the value $p = 1/n$ is used. Thus, the expected number of flipping bits equals $n \cdot (1/n) = 1$. For this n random experiments are performed. This is not very efficient.

Knuth (The Art of Programming, Volume II, 1969) pointed out that there is a much more efficient way. Instead of randomly deciding if a bit is flipped one decides randomly about the position of the next flipping bit.

It works like this: Our current bit string has length n. We have a current mutation location l. If we have $l > n$ the mutation of the current bit string is complete. We set l to $l - n$ and continue with the next bit string. If we have $l \leq n$ we mutate the l-th bit in the current bit string. We create a new current mutation location by first generating a uniformly distributed random number $r \in [0, 1)$. The position of the next flipping bit is computed by adding $\lceil (\log r)/\log(1-p) \rceil$ to l. This is the new value of l. Now we continue. In the beginning we perform one such random experiment and set $l = \lceil (\log r)/\log(1-p) \rceil$.

Implement standard bit mutations in two different ways: once following Knuth as above, once in the naive way where you perform one random experiment for each bit. Compare the actual run times for both implementations by performing 1000 mutations with $p = 1/n$ for different values of n. Start with $n = 10$ and double the value of n until n exceeds 2 000 000. Report the actual run times for both implementations.

Remark This explains why Knuth's idea is actually correct and is here only for information. Let $r \in [0, 1]$ be the random number. We have $\text{Prob}(r \leq a) = a$ and $\text{Prob}(r \in [a, b]) = b - a$. We know that the next mutation occurs at a distance of exactly i with probability $(1-p)^{i-1}p$. We observe that $(1-p)^{i-1}p = (1-p)^{i-1}(1-(1-p)) = (1-p)^{i-1} - (1-p)^i$ holds. Thus, $(1-p)^{i-1}p$ is the probability that $r \in [(1-p)^i, (1-p)^{i-1}]$. This is equivalent to $\log r \in [i\log(1-p), (i-1)\log(1-p)]$ and also equivalent to $(\log r)/(\log(1-p)) \in [i-1, i]$. This, in term is equivalent to $\lceil (\log r)/\log(1-p) \rceil = i$ and exactly what we use.

Assignments are handed out on each Friday during the lecture. Written solutions are due the next Wednesday. Feedback is given and solutions are discussed the Wednesday after that.