Problem Solving in Computer Science: Week 3

Scribe on duty: Horatiu Jula

October 7, 2008

1 Introduction

Given the currently explored permutation of the random nodes $f = \langle r_0, ..., r_m \rangle$, we compute the values $x_0^*, ..., x_m^*$ for the random nodes. If permutation f is not *self-consistent (x_i^* not monotonic) and progressive* [1], then the *next* function should choose a next permutation which may have bigger chances to be self consistent and progressive.

2 Ideas of defining *next* function

Idea 1

In the $first$ function, filter out all permutations $f = \langle r_0, ..., r_m \rangle$ that have no connections to the *max* sink. In the *next* function, skip permutations which are not progressive. Explore permutations in lexicographic order.

Lexicographic order

Permutation f is before permutation g in lexicographic order iff for the first index i where $f_i \neq g_i$ we have that $f_i < g_i$. For the algorithm for finding the next permutation in lexicographic order, see [2].

Idea 2

Sort the permutation $f = \langle r_0, ..., r_m \rangle$ according to the values $x_0^*, ..., x_m^*$. Let the sorted permutation be f'. We define the next permutation as: $next(f) = f'$. If $next(f)$ is already explored, choose another unexplored permutation in lexicographic order.

Idea 3

Let P be the set of permutations of random nodes, with p a permutation from P of the form $p = \langle r_0, ..., r_m \rangle$.

We define a histogram $h : V_R \times V_R \to \mathbb{N}$ for pairs of random nodes, with values in $\mathbb{N}$. This histogram is constructed step by step as different permutations are examined, such that after n seen permutations we have $h^n(r_i, r_j)$ equal to the number of times $x_i^* < x_j^*$ for the n explored permutations. Here x_i^* and x_j^* refer to the random nodes r_i and r_j and not the i and j positions in the permutation. Formally, the histogram at step $n+1$ will be given by the formula:

$$h^{n+1}(r_i, r_j) = \begin{cases} h^n(r_i, r_j) + 1, & \text{if } x_i^* < x_j^* \\ h^n(r_i, r_j), & \text{if } x_i^* \geq x_j^* \end{cases}, \text{ and } h^0(r_i, r_j) = 0,$$

where $i, j \in \{0, \ldots, m\}$ and $i \neq j$.

Permutation exploration in our $next()$ function at step n is done by taking into account the current permutation p and the histogram h^{n-1}. Nodes from p for which the self-consistency condition is not fulfilled ($x_i^* > x_j^*$ for $i < j$) are swapped. Should more than one pair of nodes meet this criterion, we choose the pair for which the absolute value of the difference of their histograms (both orderings taken into account) is the smallest. In other words, $|h^n(r_i, r_j) - h^n(r_j, r_i)|$ must be minimal over all $\langle r_i, r_j \rangle$ pairs that break the self consistency. Should multiple pairs have the same difference, a comparison of the maximum histogram values may be taken into account as further criterion. That is, we choose the pair for which $\max(h^n(r_i, r_j), h^n(r_j, r_i))$ is minimal. If this procedure still yields more than one pair of nodes, we randomly choose one pair. Intuitively, this mechanism encourages swapping nodes for which an order preference has not yet clearly been observed over past permutations.

If swapping two nodes will take us to an already visited permutation p' we may find ourselves in a looping condition. In consequence, the $next()$ function will choose the first unvisited permutation in lexicographical order, starting from p'.

Idea 4

In general, the Markov chain corresponding to f does not have a solution. It only has a solution for Condon SSG graphs [3][4], i.e., for SSG graphs where nodes have 2 successors and probabilities are 1/2 for random nodes.

Given the current permutation $f = \langle r_0, ..., r_m \rangle$, pick a random node r_i and move the node in the permutation on the other $n-1$ positions. From the $n-1$ resulting permutations, choose the best one (the closest to be self-consistent and progressive) to be $next(f)$.

Idea 5

Rewrite the strategy improvement algorithm in terms of exploration of random permutations, in order to use strategy improvement to improve permutations.

Challenge: how to map the improved strategy back to the permutations, i.e., how to get a permutation from a strategy. Open question: would this improve the algorithm ?

Idea 6

Construct sets W_m as follows: W_m represents the nodes that have direct edges to the max sink, W_{m-1} the nodes which are at 2-hops distance to the max sink, and so on. Generate $next$ permutation $f' = \langle r_0, ..., r_m \rangle$ based on the W_i membership, i.e., $r_i \in W_k$ and $r_j \in W_{k-1}$ implies that r_i occurs after r_j in the permutation, because it is closer to the max sink. Since the sets W_m do not change, the transitions to $next(f)$ have to be done through local improvements in the permutation f.

Question: how is completeness (all permutations explored) ensured, since only local improvements are performed?

Idea 7

In $first$ function, rank the random nodes according to their successors, to establish an initial partial order between the random nodes. After each iteration refine the partial order and build $next(f)$ such that it reflects the new partial order. In a way, this idea is similar to idea 3.

Idea 8

Perform only local improvements in the current permutation f, by swapping only a few random nodes.

3 TODO

1. Define exactly (formally) the algorithm for *next* by Thursday, Oct 2.
2. Prepare the implementation(s) for a trial run (Tuesday, Oct 7). The real contest will be on Thursday, Oct 7.
3. Finish the project report by Thursday, Oct 7. The report should contain: (1) exact description of the algorithm, (2) description of the implementation, (3) theorems or at least examples showing when the algorithm does well (polynomial time), (3) theorems or at least examples showing when the algorithm does bad (exponential time).

References

[1] H. Gimbert and F. Horn. *Simple stochastic games with few random vertices are easy to solve.*

[2] `http://www.cut-the-knot.org/do_you_know/AllPerm.shtml`

[3] A. Condon. *The complexity of stochastic games.*

[4] A. Condon. *On algorithms for simple stochastic games.*