Project Proposal

Benjamin Sorenson

November 13, 2015

1 Team Members

I will be working alone for this project.

2 Problem Description

I work in large-scale K-12 assessment, and as part of test security, our test-development team produces several scrambled versions of the same test. In order to minimize the impact of the scrambling on the difficulty of the items (test question tend to appear more difficult the later they appear in the test), our test development department is instructed to shuffle only blocks of seven to eight adjacent questions, and come up with four or possibly more acceptable scramble patterns for each block. Once this has been done, the scrambled blocks of items are handed off to me, and it is my responsibility to assemble them into a specified number of test forms (usually 8) according to the following constraints:

1. One form must contain no scrambled blocks.
2. No two forms can have more than a specified number of shared block scrambles.
3. No two forms can have more than a specified number of answer key overlaps.
4. No new sequences of a specified number of the same answer key in a row can be created by the combination of scrambled blocks.
5. Every block scramble pattern must be used at least once.
6. No block scramble pattern can be used more than a user-specified maximum number of times.

3 Description of Software

This problem is currently solved by something like randomized DFS with backtracking, implemented in a Python program I wrote. In a addition to the constraints mentioned above, the user can input parameters governing the behavior of the algorithm—the number of iterations before backtracking, and the maximum number of restarts before giving up completely (a restart is defined as backtracking all the way back to the root).

I say it's *something* like DFS with backtracking because the algorithm currently does not keep track of nodes it has visited. Instead, it relies on the fact that, because it makes a random

choice of each available node and the branching factor is s^p where s is the number of scramble patterns per block and p is the number of blocks (assuming that each block has the same number of scramble patterns), it is unlikely that it would encounter the same node more than once. Also, the backtracking happens after a user-specified number of iterations (1000 by default)—each iteration consists of randomly selecting a node at the current depth checking to see if it meets the specified constraints—if it does, add it to list of forms that meet the constraints and continue searching, otherwise discard it and try another likely different, but maybe the same node. The first time it's not successful at finding a solution, it backs up one level and selects a likely different but maybe the same node and tries again, the second time it fails, it backs up two levels, and so on until it gets all the way back to the root. When it gets all the way back to the root, this is counted as a "restart". after a user-specified number of restarts, the algorithm gives up, and returns what it has.

4 Goal

The goal for this project would be to improve the current algorithm. One of the major downsides to the current solution is that we never know if we couldn't find a solution because it didn't exist or because we gave up to early. So, I would like to add an additional check to see that the constraints are indeed satisfiable.

In addition, I think this problem could be drastically improved by using a depth-limited (because we know how many forms we want) BFS with backtracking instead of the current algorithm. I would like to compare the current algorithm to a vanilla depth-limited BFS with backtracking and a depth-limited BFS with backtracking with a most-constrained variable, least-constrained value heuristic. The algorithms would be evaluated on their average running time when a solution is possible, and how quickly they terminate when a solution is not possible.