# ECOO 2013

## Programming Contest Solutions and Notes, Part 2

Final Competition (Round 3)

May 11, 2013

**Sam Scott, Sheridan College (sam.scott@sheridancollege.ca)**

# Mutant Children: An Efficient Solution

If two children differ only by one place in one of their crossover points, they can only differ in mutation count by 1 at the most. So a better solution than the default described in the solutions document would be to take advantage of the similarity between children. One way to do this is to start with a "base mutation count" for each parent (i.e. the number of differences between that parent and the target child) and then increase or decrease these counts if necessary as you change the crossover points, keeping track of the minimum as you go.

This approach is a little tricky to get exactly right, but it is much faster – it runs in time proportional to $n^2$ instead of $n^3$, and that's a big difference when n=10 000!

# Tour de Force: Dynamic Programming

With really big problems, backtracking bogs down and won't finish before the end of the universe. So you need a different approach. The technique of memorization used in Dynamic Programming works well here. The basic idea is to keep an array of partial solutions (usually called the "memo" array). Whenever you find a solution to part of the problem, store it in the array. When it is time to recurse, check your memo array first to see if you've already solved this sub-problem. If you have, just return the value you stored in the memo array.

The tricky part is figuring out what to store in the array. One approach that works well here is to have a two dimensional array. If you have previously solved `x` cards in a row with no mistakes and you are now looking at card `y`, store the result of the recursive calls in `memo[x][y]`. Then at each step in the recursion, look to see if there is anything stored in `memo[x][y]`. If there is something there, use that number. If not, do the recursion and store the result in `memo[x][y]`.