Wizards

By Chris Jereza, Max Buchan, Cathy Pham-Le

Main Idea:

For our algorithm, we implemented a Simulated Annealing algorithm to conduct a partially-randomized local search of the solution space (orderings of wizards) for this problem. Simulated annealing is derived originally from the process of annealing in metallurgy, which is a technique involving heating and controlled cooling of materials to maximize the size of its crystals and reduce the amount of defects. Similarly, in Simulated Annealing we set a temperature variable *T* and slowly decrease it by a factor of α (typically between 0.9 and 0.99) until we find an optimal solution. We chose this algorithm because it is a relatively efficient method of optimization that avoids the deceit of local optima. Each time the main function *anneal*() is called, the program first shuffles the ordering of the wizards. It then continuously samples *neighbor* orderings (possible solutions) that are neighbors of the current ordering, calculates the cost of each sample using the *cost(solution, constraints)* function, then accepts the sample (using it as the current potential solution) with probability given by *acceptance_probability(old_cost, new_cost, T)*.

We define a *neighbor* of an ordering as the result produced by randomly moving one wizard in the ordering to a random position, we define the *cost* of a sample ordering as the number of constraints that it fails to satisfy, and we define the *acceptance_probability* of an ordering as 1 if its cost is more optimal than the current ordering cost or as *math.exp((old_cost - new_cost) / T)*, where *T* is an exponentially-decreasing “temperature” value. Briefly, we move from one ordering to another random solution if it is more optimal, but also move to a less-optimal solution with exponentially-diminishing probability. Because our “temperature” strictly decreases, we will eventually accept fewer and fewer “worse” orderings and converge on an optimum (which is hopefully a global optimum). This allows the algorithm to both explore and exploit, with a slow convergence towards exploitation. Accuracy and runtime were most greatly affected by altering the values for α and the number of iterations for the while loop. When α approaches 1, the runtime increases significantly. Similarly, when the number of while loop iterations is increased, the runtime also increases greatly. When α is closer to 0 and/or the number of while loop iterations is decreased, we observed that the runtime was faster but significantly less constraints were satisfied.

Sources

1. The Simulated Annealing Algorithm by Katrina Ellison Geltman: http://katrinaeg.com/simulated-annealing.html
2. Simulated Annealing: http://mathworld.wolfram.com/SimulatedAnnealing.html
3. What is Simulated Annealing?: https://www.mathworks.com/help/gads/what-is-simulated-annealing.html?requestedDomain=www.mathworks.com
4. Simulated annealing: https://en.wikipedia.org/wiki/Simulated_annealing