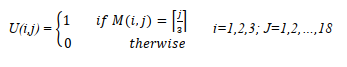
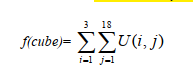
**Simulated Annealing**

**Instructions:**

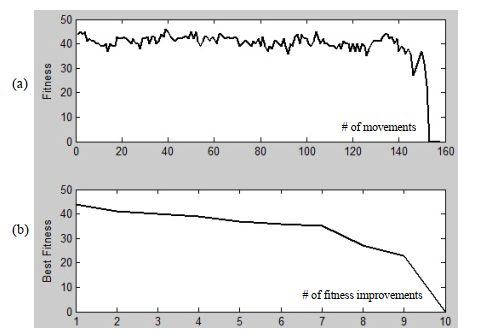
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Source 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(needs rewriting in own words!)

<https://www.ics.uci.edu/~rickl/courses/cs-271/2011-fq-cs271/2011-fq-cs271-lecture-slides/ReviewSearch271-sq2010-6%20%5BCompatibility%20Mode%5D.pdf>

* Upper bound for number of movements needed for solving the cube. This bound was first reported as 27 which were next decreased to 22 movements. It should be mentioned that these values are obtained by empirical experiments without using a scientific or mathematical calculation.
* Results show that the GA (Genetic algorithm) based approach lead to solutions with fewer numbers of movements than that of SA, although the computation time for finding such solutions is higher than SA. Considering the time aspect, the SA based approach solves the cube faster than GA approach, but offers more number of movements.
* Classic algorithms for solving the cube which mostly complete the cube layer by layer. These algorithms are hard to learn and remember the sequence of the movements, especially for a large size NNN (extended versions) of the Rubik’s cube
* eighteen basic operators which can be applied on the Rubik’s cube, rotate one layer out of 9 layers of the cube (three layers at each dimension), clockwise or counter clockwise
* Simulated Annealing approach is a random search methods
* each of the eighteen operators mentioned in previous section, makes some change on the current cube and generates a new cube, which can be considered as a neighbour of the current solution
* The fitness function in the proposed algorithm is defined as total number of small coloured pieces of all the faces, which are not located at their correct position. Thus, the problem is formulated as a minimization combinatorial problem in which, the fitness of the final solution (solved cube) is equal to zero. By defining binary matrix Uij [3× 18] as: 

So, the proposed fitness function for a given cube (or equivalent *Mij*) is defined as equation 3. 

* worst case, the fitness will be 54-6=48
* middle square of each face, is always located at the correct position and will never change.

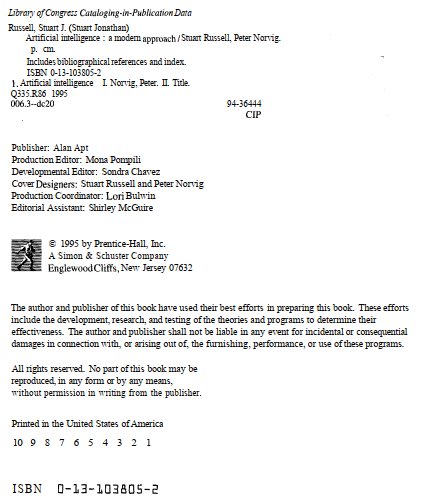


* Besides, one movement can move some pieces (maximum 9 pieces) to their correct position and lead to salient decrement of the fitness. could this be an argument for my proposed column / row check?
* the SA approach performs faster than the GA

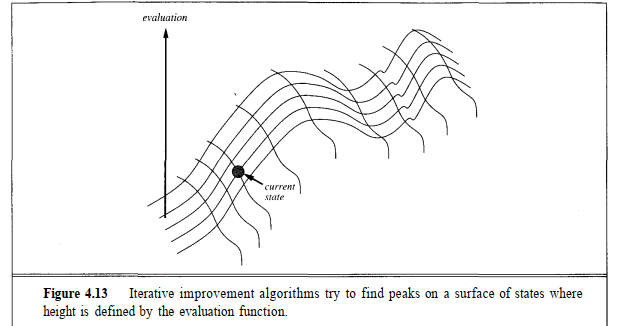
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End of Source 1\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Source 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (needs rewriting in own words!)

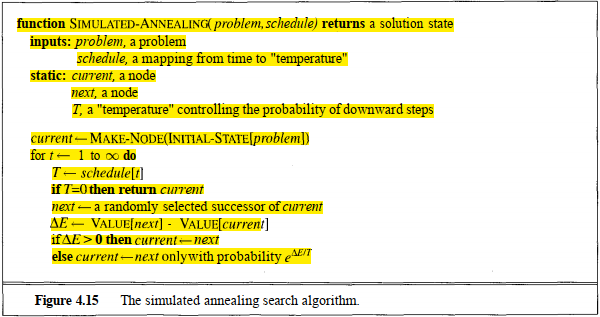
Artificial Intelligence A Modern Approach – Stuart J. Russel and Peter Norvig



**Iterative improvement algorithms:**

* Problems where the state description itself contains all the information needed for a solution and the path by which the solution is reached is irrelevant can benefit from iterative improvement algorithms. You start with a complete configuration and make modifications to improve its quality.
* Iterative improvement algorithms usually keep track of only the current state and do not look ahead beyond the immediate neighbours of the state.
* Simulated annealing algorithms can sometimes make changes that make things worst at least temporarily with a view to escaping local maximums
* Local maxima: a local maximum, as opposed to a global maximum, is a peak that is lower than the highest peak in the state space. Once on a local maximum, the , the algorithm will halt  
  even though the solution may be far from satisfactory.
* Plataux: a plateau is an area of the state space where the evaluation function is essentially  
  flat. The search will conduct a random walk
* 

\*Simulated annealing when on a local maximum allows the search to take some downhill steps to escape the local maximum.

* The innermost loop of simulated annealing is quite similar to hill-climbing, instead of picking the best move, however, it picks a random move. If the move actually improves the situation it is always executed. Otherwise, the algorithm makes the move  
  with some probability less than 1. The probability decreases exponentially with the "badness" of the move—the amount AE by which the evaluation is worsened.
* At higher values of T, "bad" moves are more likely to be allowed. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. The schedule input determines the value of T as a function of how many cycles already have been completed.
* History: The reader by now may have guessed that the name "simulated annealing" and the parameter names A£ and T were chosen for a good reason. The algorithm was developed from an explicit analogy with annealing—the process of gradually cooling a liquid until it freezes. The VALUE function corresponds to the total energy of the atoms in the material, and T corresponds to the temperature
* *schedule* determines the rate at which the temperature is lowered if *schedule* lowers *T*slowly enough, the algorithm will, find a global optimum.
* Iterative improvement algorithms keep only a single state in memory.
* For constraint satisfaction problems, variable and value ordering heuristics can provide huge performance gains. Current algorithms often solve very large problems very quickly.  
  

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_End Source 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Source 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(needs rewriting in own words!)

Metaheuristics can Solve Sudoku Puzzles

* The resultant search space is therefore:



(where f(r,c) is a function that indicates how many cells in square, are non-fixed)

This wont be the same for rubix cube because we don’t restric cells unless we were to consider central face elements as fixed but I believe in our case we need a slight alteration on function

* Probability calculated using:



* where d is the proposed change in the cost function and *t* is a control parameter, known as the  
  temperature
* the way in which *t* is altered during the run is of great importance to the success and  
  expense of the SA algorithm
* *t*0 should allow the  
  majority of moves to be accepted and slowly reduced (according to a *cooling schedule*) algorithm becomes increasingly greedy
* *t*0 should be chosen with care: a value that is too high will mean that computation time is  
  wasted because the algorithm is likely to conduct an unhelpful random walk through the search space.  
  On the other hand, a value for *t*0 that is too low will also have a negative impact, as it will make the  
  search too greedy from the outset, and therefore make it more susceptible to getting stuck in local  
  minima.
* *t*0 should allow approximately 80% of proposed moves  
  to be accepted can be achieved by measuring the variance in cost for a  
  small sample of neighbourhood moves.
* The initial temperature *t*0 is simply the standard deviation of the cost during these moves.
* Termperature is reduced with geometric cooling schedule where the current temperature *ti* is modified to a new temperature *ti*+1 via the formula:
* 
* a is a control parameter known as the *cooling rate*, and 0 < a < 1.
* Obviously, a large value for a, such as 0.999, will cause the temperature to drop very slowly, whilst a setting of, say, 0.5 will cause a much quicker cooling. In this approach we also use the version of SA known as the *homogenous* SA scheme. This basically means that the algorithm takes the form of a sequence of Markov Chains, where each Markov chain is generated at a fixed value of *t*, and *t* is then altered in-between subsequent chains (van Laarhoven and Aarts, 1987). Again, the value defining the length of each Markov chain (here we call this value *ml*)



Note that such a value allows a good chance of each pair of non-fixed cells being considered at least once per Markov chain. Finally, we add a random-restart mechanism to the algorithm: If no improvement in cost is made for a fixed number of Markov chains (in all tests this was set to 20), then *t* is reset to its initial setting *t*0, a new initial solution is generated, and the algorithm then ontinues as normal. We term this process a *reheat*.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End of Source 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**My two cents:**

* I suspect annealing is really only applicable to a situation where number of steps aren’t evaluated, in my case the best solutions I obtained were in the regions of 2,000 steps but ranging to as much as 15,000. Source 1 provided claims that managed to average about 250 steps which is suspiciously good considering a 22step lower bound. The issue being annealing encourages taking many bad steps initially so as to avoid local minima but in doing this compounds the step count. So if this was compared against other algorithms it would be a no brainer that this process wouldn’t be most step friendly.

I considered the idea of a backtrack mechanism to allow rejection of steps in the hopes of reducing step counts, as it stands the algorithm can take a step left followed by a step right resulting in two unnecessary step counts and no gain. I decided against this because it would be mudding annealing process. The idea was to apply annealing and compare this algorithm so I decided to try keep it as close to the documented process as possible.

* Another consideration is that typically this search is done/ or visually demonstrated with a continuous function graphic with lots of nice rolling hills, but the reality is if your near completion on a rubix cube moving a single row will destroy your cost value (particularly if evaluating based on combined rows and columns ?). Essentially non-invertible, Visualised:

Because our cost function provides a discretised cost evaluation, the possibility of a delta zero occurrence becomes highly likely which would result in a probability of 1 and never could be accepted. The role of the cost function seems very significant in theory however I couldn’t compare with differing algorithms in practice

* The current solving situation involves the concept of tempering where if no improvement is made we use reset cube to initial temperature to allow more unfavourable moves.
* I thought I could save on the moves by resetting the initial conditions and move counter after tempering as we are increasing the bad decision moves possibly to such an extent that the cube is equally jumbled as it was originally? (I tried this hypothesis and cant definitively say but it didn’t appear to work, work)
* Delta\_Cost <= 0, this was an issue encountered because states of the same value will cause an issue when calculating probability. When you look at annealing books and examples I think mostly they are applied to relatively nice functions that arnt so discrete, because of this the event of a Delta\_Cost=0 never likely. But in our case that’s a whole extra move for in theory no benefit so really it should be evaluated in the negative move scenario but e^0 =1 always so would always either be accepted or rejected.
* Parameters seem way too finicky, the practical solution doesn’t seem to work with theoretical values suggested to a great extent. Initial probability accepted at t0 is meant to be roughly 80% but I believe with theoretical calculations this looks to be nearer 99%. I believe current implementation is depending much less on the idea of annealing than it should but I simply can’t tweak the values to replicate the values source 1 obtained. They don’t provide source code.