# 5 Local Search and Optimization Problems

The problem covered so far dealt with agents having to choose a series of actions to reach the goal state, where the path to the goal also constitutes the solution to the problem. In some other problems, the path to the goal is irrelevant as only the end state is required for the solution.

An example would the **n-Queen puzzle**. Given an $n \times n$ gird board, and $n$ number of queens which can travel horizontally and diagonally to attack, what is the configuration of the queens on the board so that none of the queens can attack each other in one move?

Suppose we are given the $4 \times 4$ chess board on the left of Figure 14. The initial board is clearly not the solution because each queen can attack other queens diagonally. How can we design and algorithm that will lead to the solution board on the right?

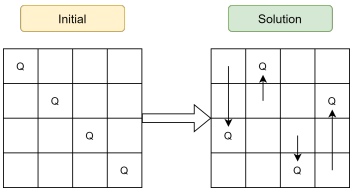


Figure 14: $4 \times 4$ Board with $n$ Queen

One way to break down the problem is to restrict the transition of one board state to another to the movement of queens across rows (i.e. along the columns) because we already have one queen in each column. This means board can transition to another state by moving up and down. The new abstracted problem can be broken down as follows:

1. Board States $S$: Represents a state of the board with queens at various positions.
2. Neighbours of a State $N(S)$: Represents the new board state brought about by an action such as moving the queen.
3. Value of State val$(S)$: The value should reflect the notion of 'quality' of a state, in a sense of how close it is to the goal state. Naturally val$(S) = 0$. For n-queen, val$(S)$ is the number of pairs of queens that can attack each other.

## 5.1 Hill-Climbing Search Algorithm

A useful algorithm to implement to find the solution is to run the Hill-Climbing Search algorithm. It is a loop that would continually move the game state to an increasing (or decreasing) value until it reaches a local 'peak'. It does not look beyond the immediate neighbouring states, and does not remember past states.

**Algorithm 16** HillClimb$(S)$

```
minVal ← val(S)
minState ← {}                                  ▷ Only 1 thing to track
for each u in N(S) do
    if val(u)< minVal then
        minVal = val(u)
        minState = u
return minState
```

To solve n-Queen, the Hill-Climbing Search algorithm can be called multiple times until a solution is found.

**Algorithm 17** SolveNQueen(*initialState*)

1: $S \leftarrow initialState$
2: **while** val($S$)!=0 **do**
3: $\quad S \leftarrow$ HILLCLIMB($S$)
4: **return** $S$

### 5.1.1 Weakness of Hill-Climbing Algorithm

One weakness of the Hill-Climbing algorithm is that it greedily chooses the neighbouring state that has a better value. However, at times there are occasions where in order to reach the solution, the state has to transition to a slightly worse value first. For example in Figure 15, the board state has to transition

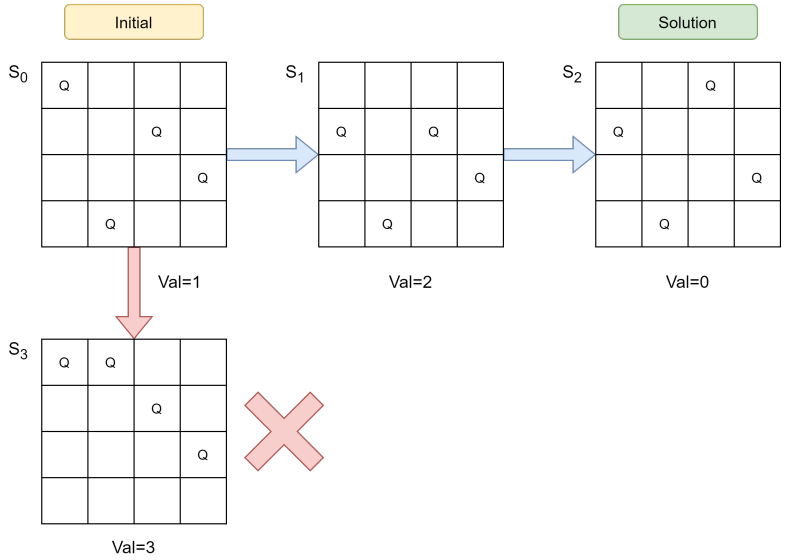


Figure 15: Board state having to transition to higher value to reach solution

from initial state $S_0$ with a value of 1 to state $S_1$ with a worse value of 2, before it can reach the solution $S_2$ with a value of 0. Thus a more ideal algorithm would allow for a state transition that worsens the state value occasionally. However, this allowance of state transition to a worse state value should be weighed against some form of preference that would lead to the right solution. As seen from a transition of initial state to state $S_3$, the value worsened to 3, but the board is further away from the solution.

## 5.2 Simulated Annealing

As we already discussed that Hill-Climbing has an issue of getting stuck at local minimum/maximum. One of possible way to over this issue is to *allow mistake* sometimes; allowing mistake is to allow to choose neighbor that have lower(resp. higher) value than the current state, and that might not be the ideal move to achieve minimum (resp. maximum). One such algorithm, that allow us to make mistake is *Simulated Annealing*. Simulated Annealing is inspired by the process in metallurgy where metal or glass is hardened by heating them to high temperature and than cooling them down gradually (based on a cooling schedule) so that the material will harden into a low-energy crystalline structure. In condensed-matter, we have a type of spin models, i.e we have atoms and all of them have spin either $+1$ or $-1$ with equal probability. Let us assume $\mu_i$ is the spin of atom $i$, then the $Pr[\mu_i = 1] = \frac{1}{2}$, and the energy of the system $C$ is given by:

$$E(C) = e^{-\frac{\sum\limits_{(i,j)\in N} J\mu_i\mu_j}{K_B T}}$$

where $T$ is the temperature and $K_B$ is the Boltzman constant. At normal temperature, we have on an average half of the atoms labeled $-1$, and another half as $+1$. Our objective here is to achieve a state with lowest energy. A state $C$ is at lowest energy, if all the atoms at state $C$ have spin $+1$ or $-1$. The probability of moving a state $C_a$ to $C_b$:

$$Pr[C_a \to C_b] \propto e^{-\frac{E(C_b) - E(C_a)}{K_B T}}$$

Now, at this point, the question is *How can we get into the lowest energy state*. The procedure is, first have the high temperature and then slowly decrease the temperature. How the material scientists facilitate the transition of a configuration to a lower energy state is through the cooling schedule. The cooling schedule is given by the function that outputs the temperature $T$, and is a decreasing function of time. As a result, at the beginning when temperature is high, the configuration easily moves with a high probability even when $E(C_b) > E(C_a)$. As time goes on and the temperature falls, the $Pr[C_a \to C_b]$ starts to decline. So basically, at the start, we allow atoms to transfer to the high energy state in order to finally achieve global minimum energy state.

Algorithm[8] 18 describe the process of simulated annealing to achieve global minimum. The *Schedule* function typically decreases with time but it is always up to the user to define the function on how gradual the simulated annealing process should be.

**Algorithm 18** SimulatedAnnealing(*initialState*)

```
1:  C ← initialState
2:  for t = 0 to ∞ do
3:      C' ← PickRandomNeighbour(C)
4:      T ← Schedule(t)
5:      if val(C') = 0 then                          ▷ Assume that val(Goal) = 0
6:          return C'
7:      if val(C') < val(C) then
8:          C ← C'                                   ▷ C' better than C
9:      else
10:         C ← C' with Probability ∝ exp{-(val(C')-val(C))/(K_B T)}   ▷ C' equal or worse than C
```

[8] Taken and modified from Khiew Zhi Kai's class scribe.

In Algorithm 18, we terminate when $Val(C') = 0$, but for some cases, the $Val(Goal)$ may not be defined. Such a problem can be seen as an optimization problem to minimize the value of solution state. We can then define the *Schedule* function to return 0 value based on certain criteria or after a finite number of iterations or modify the terminating criteria as per the application.

The idea of simulated annealing works well in practice, but it is not guaranteed to achieve global mini-ma in every case. Hence, like hill-climbing, simulated annealing is also not complete.