ALEKSANDER SKRAASTAD & FREDRIK B. TØRNVALL

EXERCISE 4

TDT4136 - Intoduction to AI

October 2014

Norges teknisk-naturvitenskapelige universitet

Table of contents

Chapter 1

Egg Carton Puzzle

1.1 Solutions

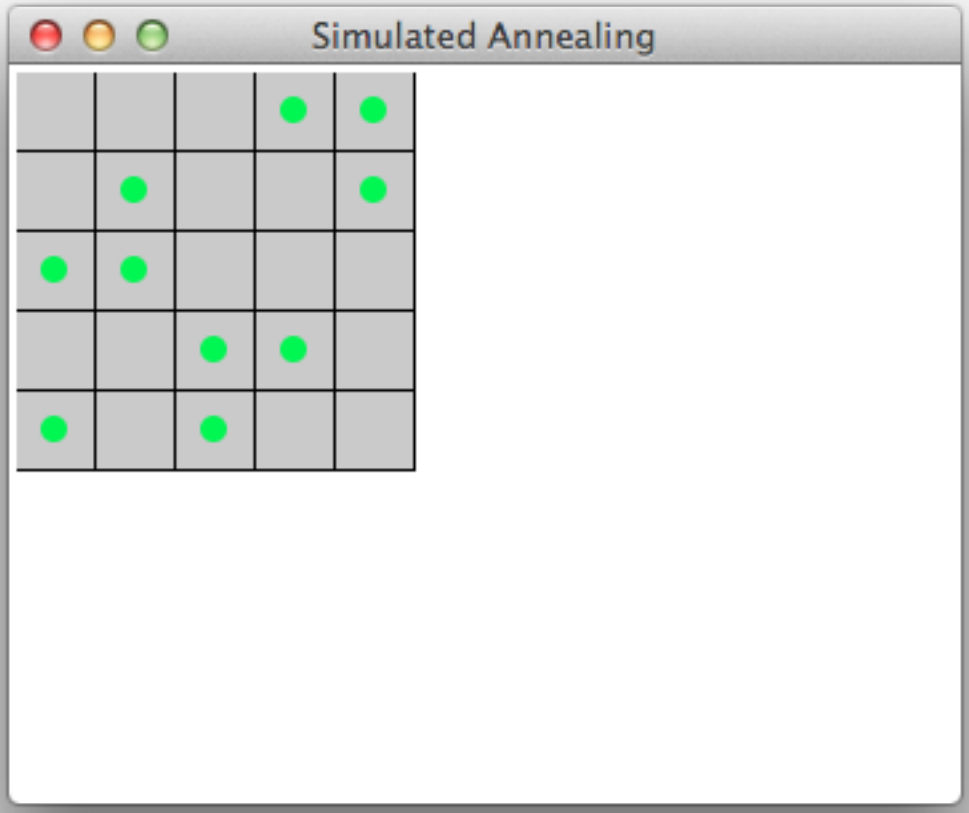
Below are solutions to the different configurations of boards. Our implementation wraps everything into a GUI program that lets you choose board first, and then run the Simulated Annealing algorithm on the board.

To represent solutions that give an optimal solution within the set temperature and decrement value, green dots are place on their respective tiles. Solutions that are sub-par but are the best at the time will be represented as a board with red dots.

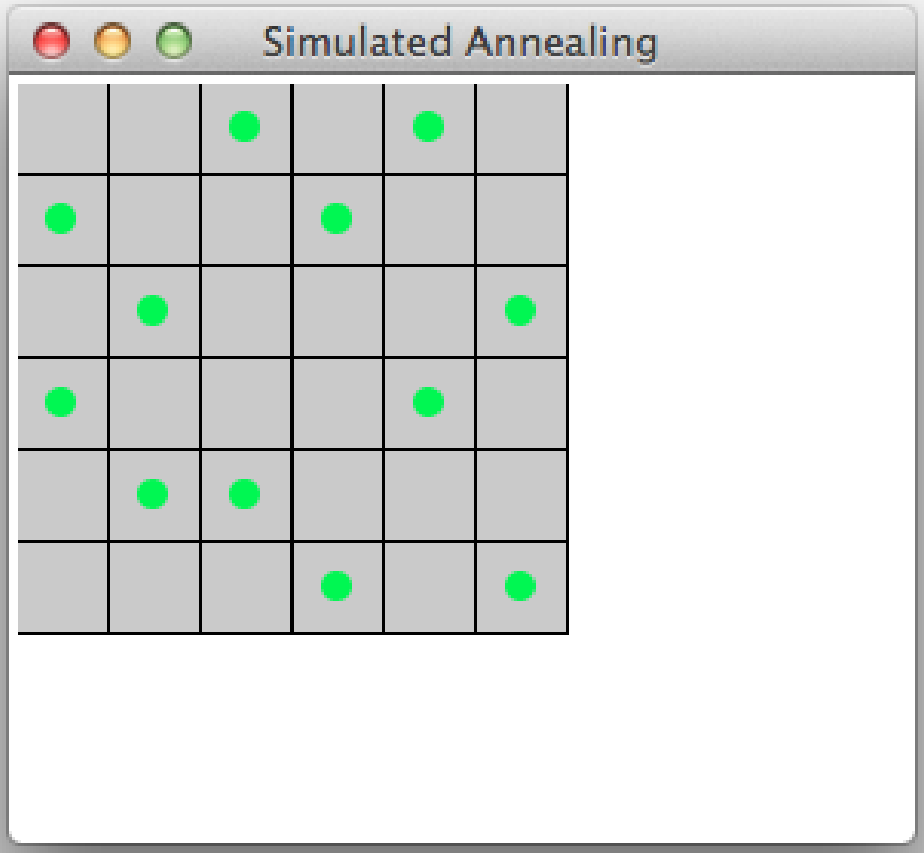
The program is written in Python using the Tkinter GUI library, and is run from $main.py$

Under some circumstances, which we failed to identify, the application goes into deadlock and does not time out yielding red dots. Instead the application hangs and must be forcefully exited and restarted.

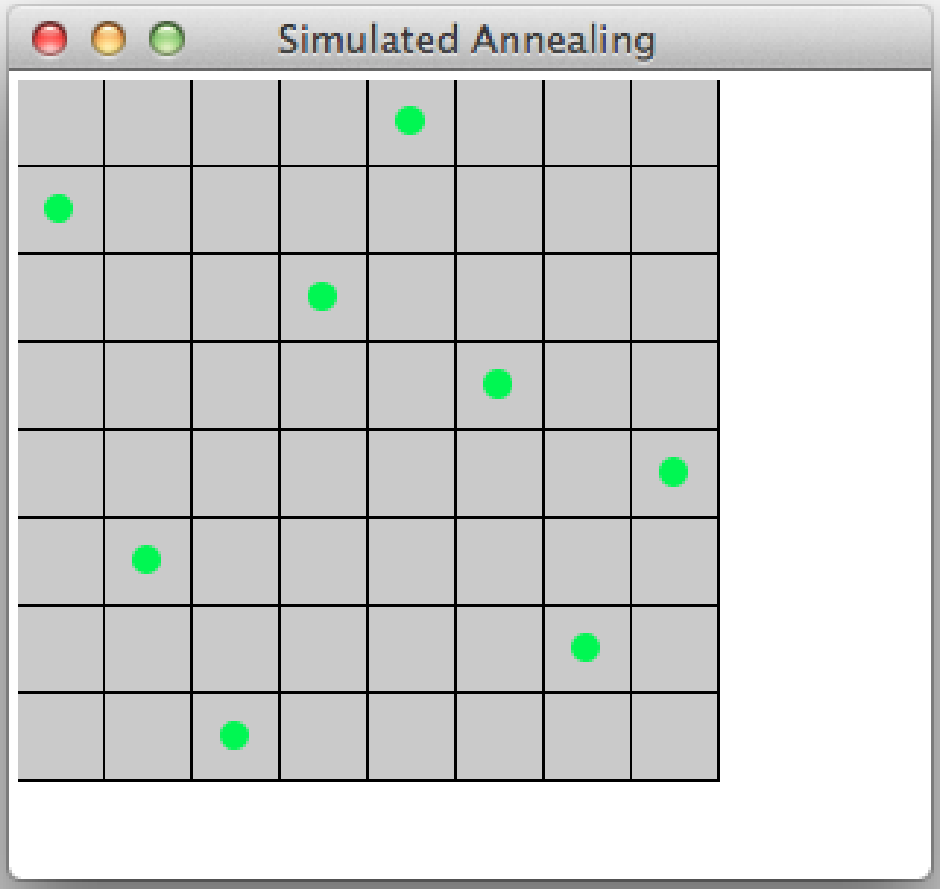
1.1.1 5x5K2 solution



1.1.2 6x6K2 solution

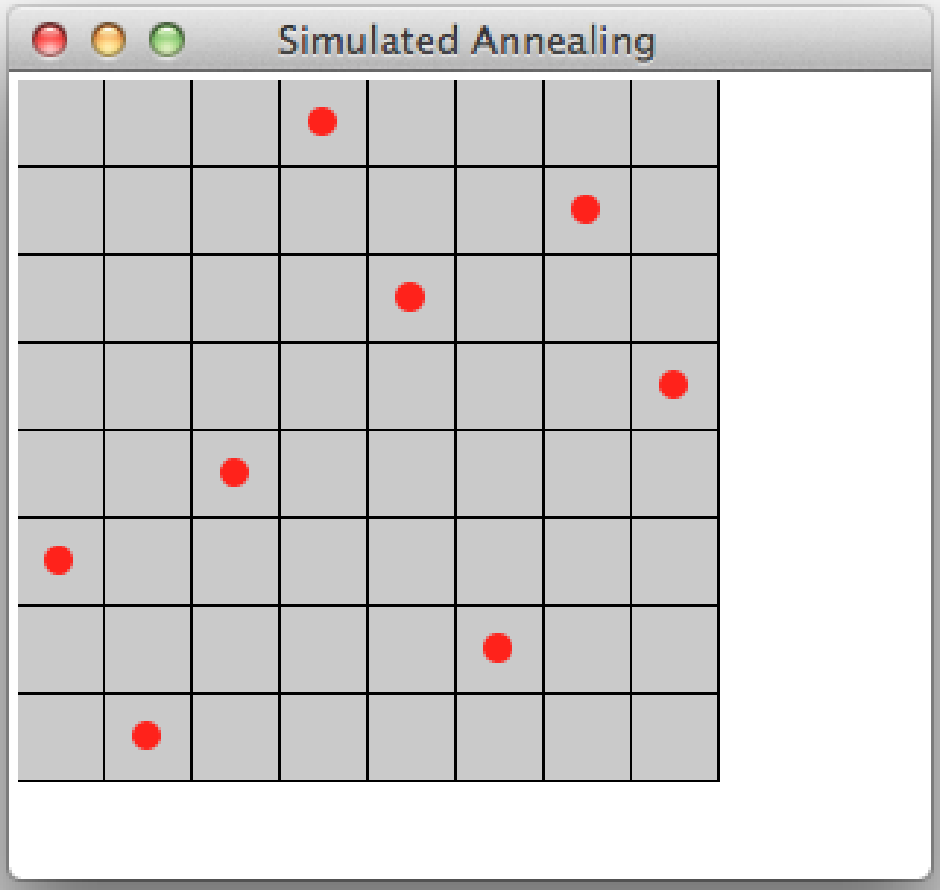


1.1.3 8x8K1 solution

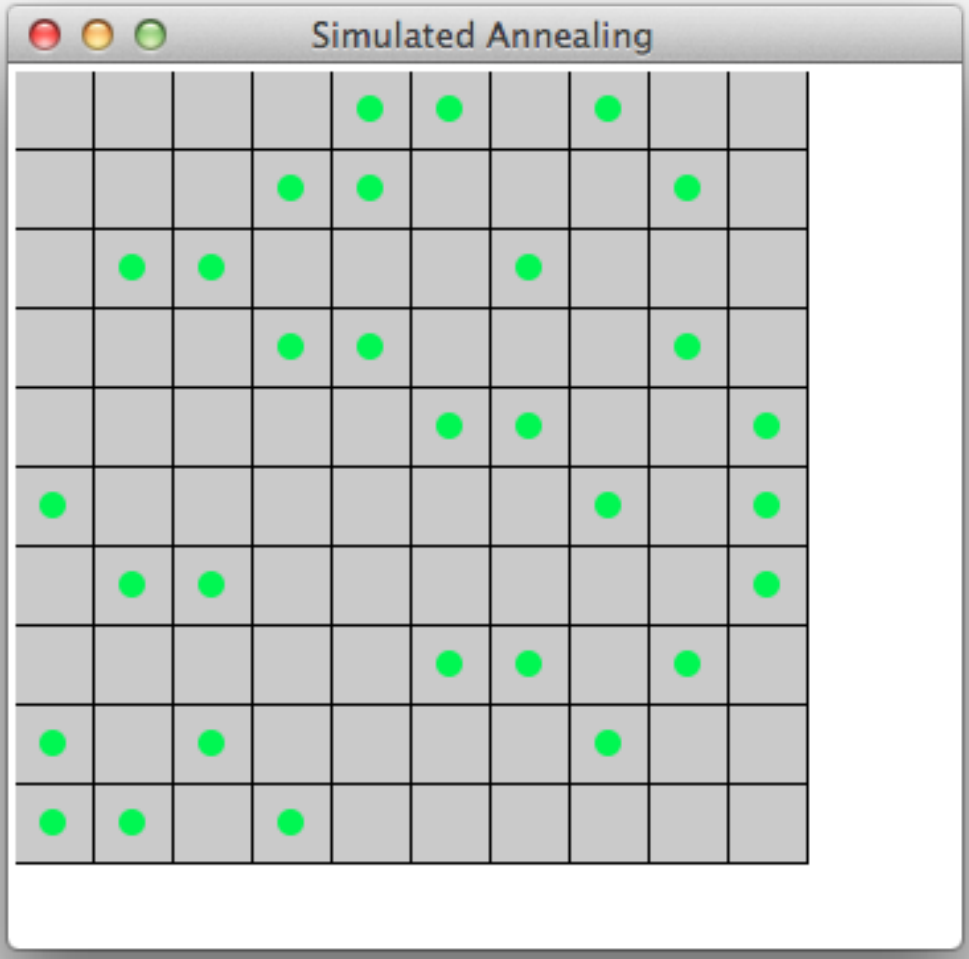


Yes this is a familiar problem, it is better known as the 8 queens puzzle.

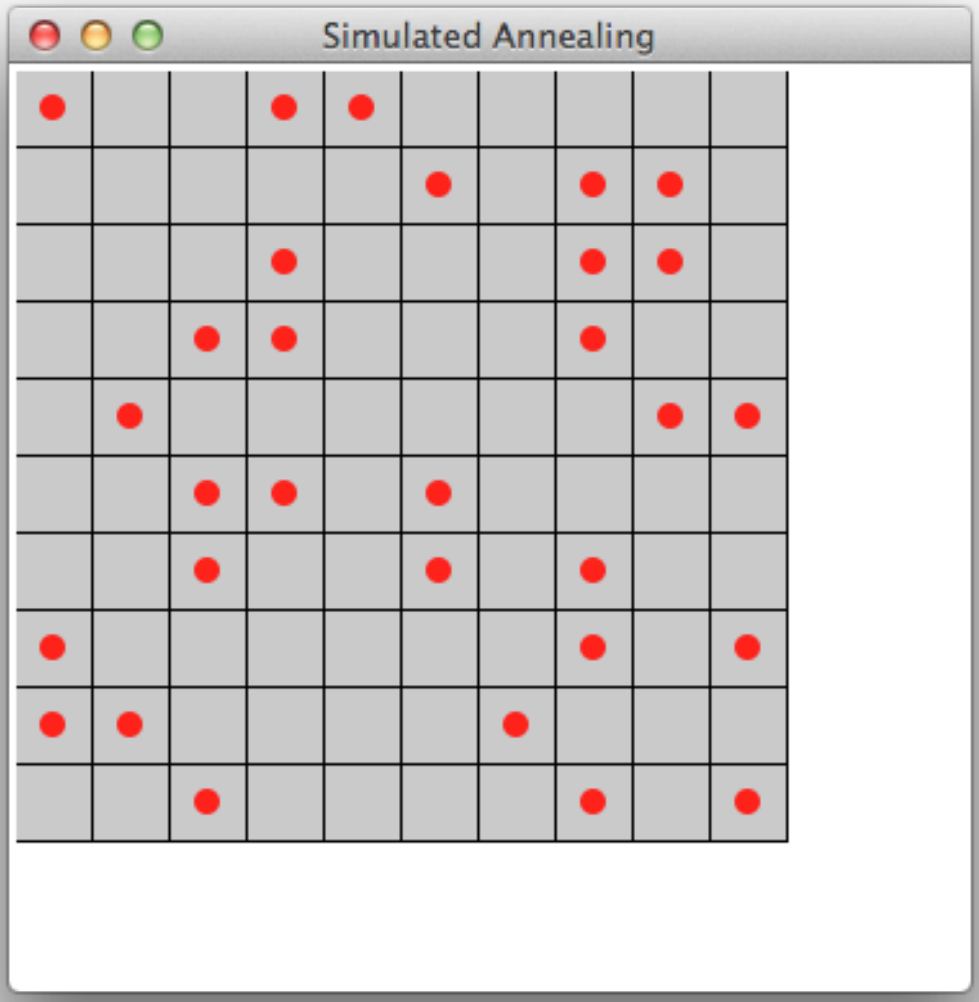
1.1.4 8x8K1 best solution when time expired



1.1.5 10x10K3 solution



1.1.6 10x10K3 best solution when time expired



1.2 Objective function

For the objective function we have used the following approach.

We have the value O, which represents the value of $F(P)$. This value is calculated using the following approach:

```
1. Concatenate rows and columns into a single list
2. Calculate the delta d(x,y) for each column or row from the K value.
3. Sum up the absolute values for each row or column in the list of deltas.
4. Add all overflowing diagonals to a new list
5. Check if there are any overflows, if not, return 1 and exit.
6. Calculate O using the following formula:

O = 1 - (rowcol_slots_available / (K * M))

7. If there are diagonal overflows, adjust O by the following value:

O =  O - (|diag_overflow| * 0.05)

8. Return O
```

This will return an $F(P)$ value that dynamically scales based on the amount of misplaced eggs on the board with a primary focus on rows and columns. For the diagonals, we have a linear scaling based on the amount of diagonals that contain overflows, with the exception of underflows, as there may potentially be many diagonals that contain available spots on optimal solutions.

This weighting of diagonals will on small boards have more weight on diagonal overflows, and focus on rows or columns on larger boards.

This $F(P)$ value will then penalize underflows and overflows equally for rows and columns, thus giving very high values for low over/underflows in rows and columns, hinting to being close to an optimal solution. Then, many of the neighbors created from this high-value board will be over/underflow free and randomness from the diagonal overflows will hopefully generate a neighbor that will allow row/col swapping to give a solution free of overflows in the diagonals.

1.3 Neighbor creation

For the neighbor creation function, we have worked towards smart generation for rows and columns, and some randomness in diagonal overflows.

When the function starts, there is a 50/50 chance of copying the board from which neighbors are generated or copying the previous neighbor.

Next, it examines the rows and attempts to move an egg from an overflowing row to an underflowing row. If this swap occurs, it adds this permutation as a neighbor and proceeds to generate the next. If there are no overflows in the rows, it does the same trick to the columns.

If there are no overflows in rows or columns, some randomness is added to the neighbor generating function. This is mainly because we had problems getting the diagonal neighbor generating error free. We had some issues with extra eggs on the board that caused the algorithm to deadlock. This is also to allow it to break free from local best peaks that are potentially not globally optimal.

It does so by a 4/10 chance to move randomly selected egg from an overflowing diagonal to a randomly selected available spot on the board. It then adds the permutation as a neighbor and proceeds. On the off chance, a totally random board is created, this is to prevent deadlocks in our implementation.

An example of a neighbor of a given board B with 3 eggs in row 4 and 1 egg in row 5 will at first move a random egg from row 4 into row 5 and add it as a neighbor. If this move causes a column to overflow, it will perform a similar operation on the columns and add it as a neighbor. This process of checking will continue until we have n neighbors, either by row and column swapping or some added randomness from the diagonal overflows.