Report practicum 1

Sunny Hsieh, Abel de Groote, Mika Sie

6534856, 6748236, 6603726

Findings for local search

In this report we describe our findings during practicum 1. In this practicum we programmed a local search algorithm. Our problem domain was a sudoku existing of 9 by 9 squares, grouped in tiles of 3 by 3 squares. In every row and column each number from 1-9 can only be found once, conform to the rules of sudoku. The task was to program an iterated search algorithm. This algorithm works in the following way:

1. One of the 3 by 3 tiles is chosen randomly
2. It tries all possible swaps of 2 non-fixed numbers
3. Choose the best option if it has a better or equal score than before

We created this algorithm in C#. The following is the thought process and progress on how we created the algorithm.

-First we started with processing the input and creating the initial board that contains 9 rows and 9 columns.

-Next, we created a random board, where all zero numbers get a number in way that the numbers 1-9 occurs only once in each block (there are 9 blocks in sudoku), we wrote two help methods here: vulbord and vulblokje.

-Then, we implemented the hill climbing algorithm. We used multiple help methods, such as: a swap method that swaps two elements; the evaluation function method (and help method) to evaluate the state before and after swapping; best successor method to find the best successor, so this method return the best next board (if it is at least as good as the previous board). At the end hill climbing returns a local optima, a board that sticks in a plateau or a solution.

-When we have found a local optima it was necessary to implement a random walk algorithm in order to escape the local optima.

-After using hill climbing and random walk multiple times, it must have found a solution, we wrote a method to print the solution of the board in a pretty way.

When we finished coding our algorithm we worked on collecting our data. We collected the data by running the grid as input and noting the runtime it took in an excel sheet. For each grid we had four S-values, this is one of the parameters. For every S-value we ran the grid ten times so we had more reliable data. For these ten runs we calculated the average runtime so we had a good idea of the runtimes per S-value. For every average runtime we

created a table and a corresponding graph. This table and graph would ensure that our data can be better interpreted. This whole process was repeated for every separate grid.

Choices of the parameters

We have two parameters in our code:
-Plateauparameter
-S-value

Plateau parameter

For the plateau parameter we have chosen the value 18. We only saw a difference in choosing a low value of the plateau parameters (e.g. 1 or 2). It had a negative impact on the runtime. The reason for this is because it could have returned a state of board that isn't a local optima. When we choose a high value of the plateau parameter it doesn't matter anymore if the value is for example 18 or 19. The more interesting parameter is the S-value.

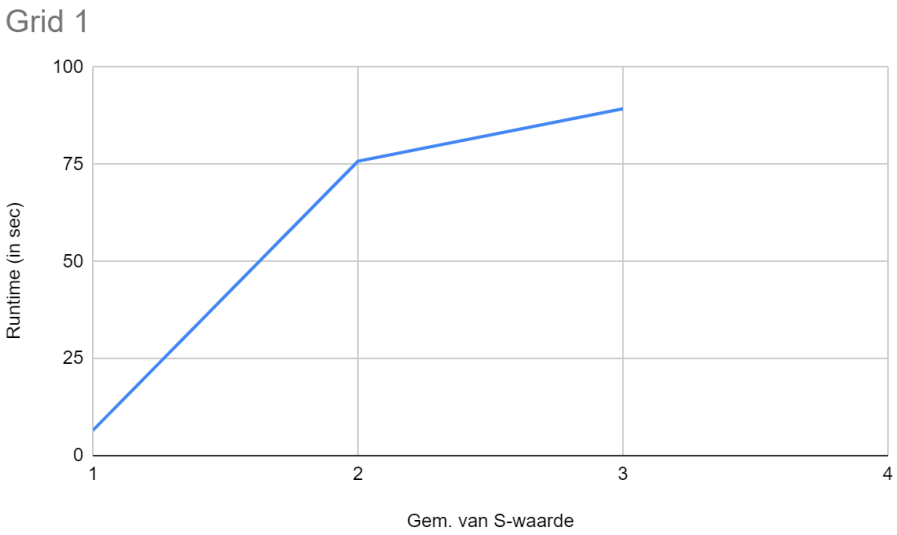
S-value

The S-value has a great impact on the runtime of finding the solution of a grid. We have run the code on each grid with a certain s-value 10 times. First with s=1, then s=2 and s=3 (except for grid 2) and s=4 only for grid 5. The average runtime of each grid corresponding to each value of s is shown below.
In some tables you can see the term "Too long". We classified a run as "Too long" when we expected the run to take longer than an hour. In the syllabus you can see that for some S-values we did run the algorithm but it took almost half an hour to an hour. We didn't think it was viable for us to let our computers run our code for that long. This is because it would simply take too much time.

Results

Grid 1

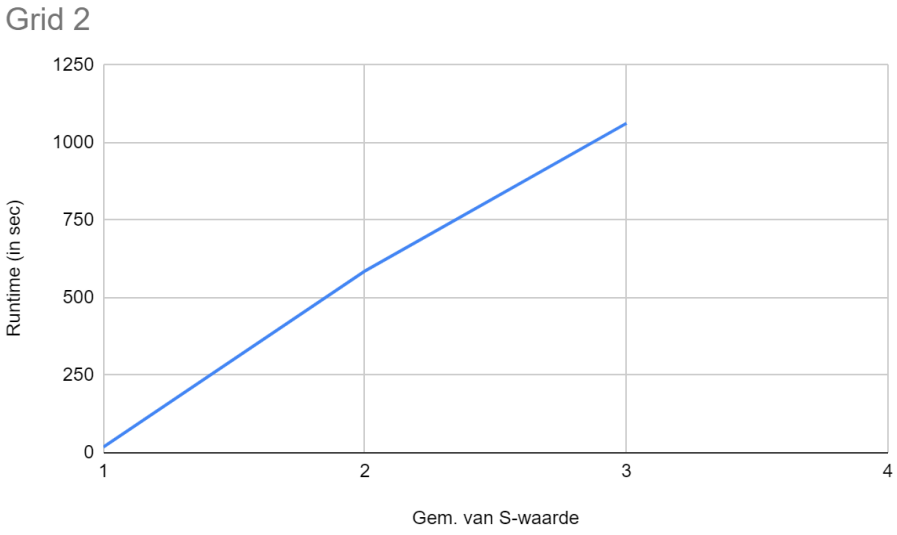
Grid 1	
S-value	Average runtime (in sec)
1	6,5
2	75,8
3	89,3
4	Too long



When the S-value is 1 we can see our algorithm has a pretty fast runtime. We can clearly see that when the S-value is increased, the runtime starts to increase drastically. The difference between 2 and 3 isn't even that big but when the S-value becomes 4, it would take too long to run the algorithm as it couldn't converge.

Grid 2

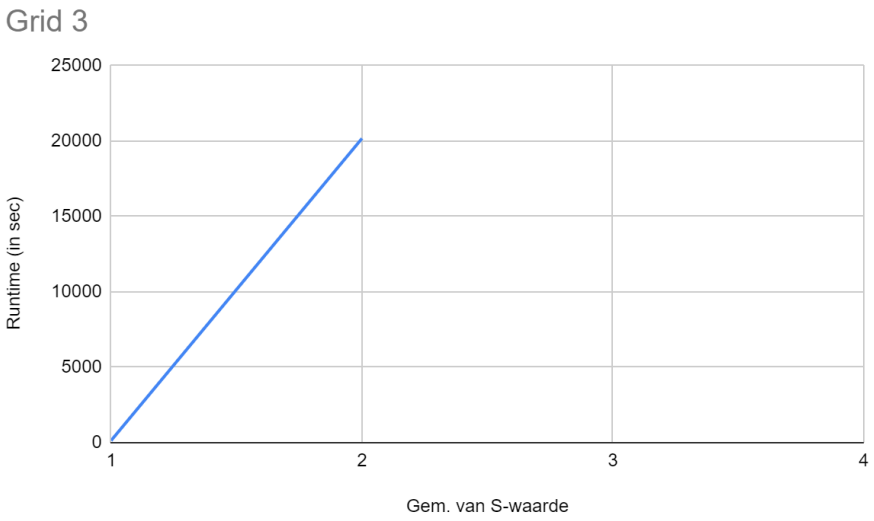
Grid 2	
S-value	Average runtime (in sec)
1	18,5
2	586,35
3	1062,4
4	Too long



Here we can see some similarities to grid 1. A S-value of 1 ensures we have a pretty okay runtime. But when the S-value becomes 2 it takes 50 times longer to come to a solution and when the S-value is 3 it is doubled compared to an S-value of 2! An S-value of 4 was again too long for us to measure.

Grid 3

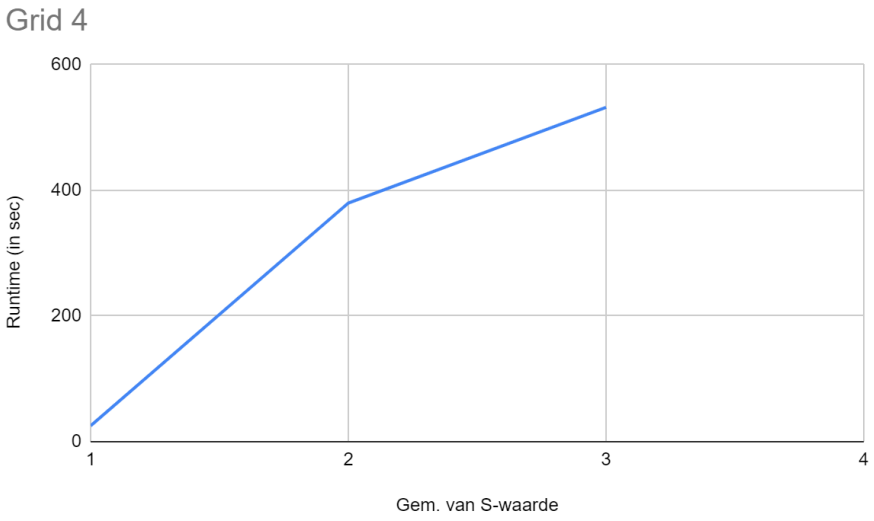
Grid 3	
S-value	Average runtime (in sec)
1	136,7
2	20173
3	Too long
4	Too long



The third grid was a hard grid to solve. We can see this in the fact that the basic S-value of 1 already took pretty long. When we went to the S-value of 2 it took a whopping 20173 seconds on average. But there is a side note to this; we only did two runs for this S-value because it took so long. After that we stopped because it took too much time and we saw that it wasn't viable for us anymore. For the S-values 3 and 4 we had the same issues.

Grid 4

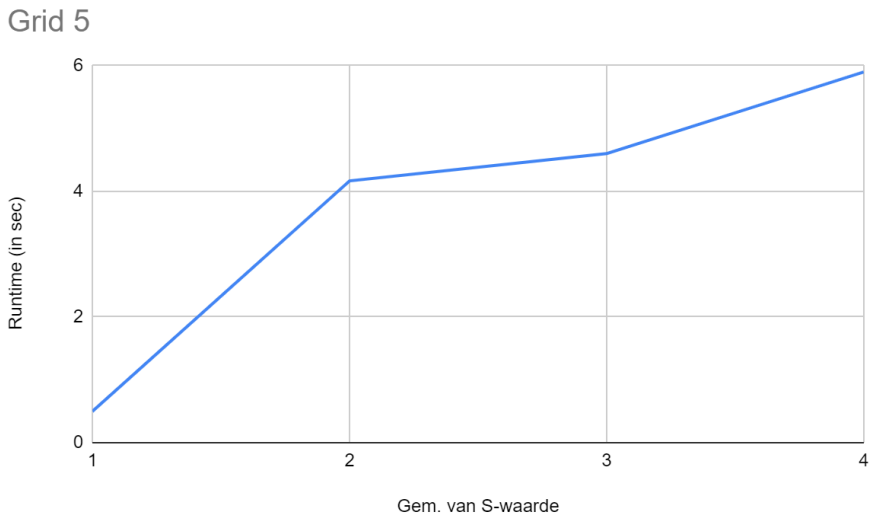
Grid 4	
S-value	Average runtime (in sec)
1	25,2
2	379,75
3	532,1
4	Too long



For an S-value of 1 the algorithm took over 25,2 seconds to converge. We see again that when the S-values are increased the average runtime gets up significantly. Again an S-value of 4 took too long.

Grid 5

Grid 5	
S-value	Average runtime (in sec)
1	0,5
2	4,166666667
3	4,6
4	5,9



Grid 5 was the only grid where the algorithm found solutions for all four S-values. But the S-value of 1 was clearly the fastest. We can see that the higher the S-value gets, the longer it takes to come to a definitive solution.

Conclusion

For each grid, they had a different average runtime for s=1. A logical explanation for this is that the suduko's had different kind of levels such as:
-very easy: grid 5
-easy: grid 1
-medium/hard: grid 4/grid 2
- very hard: grid 3

However, for each grid, we have seen a significant increase of average runtime when the S-value was enlarged. We can conclude that the higher the s-value becomes, the longer the runtime gets.

Explanation

The only unanswered question is why is it that the higher the s-value gets the longer the runtime becomes? Our explanation for this is because when you have a local optima, it has an evaluation value of (let's just call it) x. When s =1, in the best situation the value remains the same, in the worst case, x becomes x-2. But when you use the hill climbing algorithm at this point again, it will always end up in a board situation with an evaluation value <= x. Because if it can't find a state where the evaluation value is smaller than x, it would just simply swap back the 2 numbers that the random hill walking algorithm had done. Thus for s=1, it will always end up in a situation where it is at least even good or maybe even better than the previous local optima.
For s=2, in the worst case scenario, x could become x-4, and the higher the s, the worse the x can become. And for a higher s, it is not even guaranteed that you'll find a better board state with evaluation value as good as the previous value x.
Thus the smaller the s, the shorter the runtime.

Reflection on programming

The implementation of the hill climbing algorithm took several hours (around 3 hours), the only big struggle was that we had shallow copied boards instead of deep copying them. After finding this bug the hill climbing was easily fixed. For the random walk algorithm it was written within 20 minutes.

The programming went smoothly. We went to two seminars, prepared with several questions, where the TA had explained the questions for us such that we constantly had a clear vision about what we need to do next and how.

The code was finished almost 3 weeks before the deadline. Therefore we had enough time experimenting with the value of s. In the end we have runned the code a total of 142 times (see data below).

Syllabus

Grid 1

S value	Runtime (in sec)
1	5
1	7
1	4
1	0
1	3
1	4
1	10
1	7
1	18
1	7
2	36
2	34
2	84
2	26
2	16
2	3
2	27
2	121
2	66
2	210
3	19
3	44
3	138
3	207
3	85
3	131
3	79
3	108
3	28
3	54
4	Too long
4	Too long
4	Too long

4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long

Grid 2

S value	Runtime (in sec)
1	6
1	9
1	14
1	37
1	29
1	27
1	19
1	13
1	6
1	25
2	133
2	14
2	97
2	72
2	49
2	209
2	147
2	41
2	113
2	228
3	767
3	3550
3	216
3	302
3	721
3	1608

3	1290
3	1541
3	186
3	443
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long

Grid 3

S value	Runtime (in sec)
1	235
1	130
1	73
1	174
1	190
1	30
1	81
1	67
1	83
1	304
2	1719
2	38627
2	Too long
2	Too long
2	Too long
2	Too long
2	Too long
2	Too long
2	Too long

2	Too long
3	Too long
3	Too long
3	Too long
3	Too long
3	Too long
3	Too long
3	Too long
3	Too long
3	Too long
3	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long

Grid 4	
S value	**Runtime (in sec)**
1	4
1	3
1	37
1	15
1	3
1	18
1	23
1	79
1	67
1	3
2	338
2	13

2	86
2	139
2	22
2	836
2	342
2	316
2	135
2	47
3	1091
3	122
3	132
3	1039
3	52
3	724
3	1080
3	644
3	390
3	47
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long
4	Too long

Grid 5	
S value	**Runtime (in sec)**
1	1
1	0
1	0
1	2
1	0

1	0
1	0
1	1
1	1
1	0
2	4
2	2
2	1
2	2
2	2
2	1
2	2
2	0
2	4
2	2
3	9
3	4
3	4
3	4
3	1
3	18
3	1
3	3
3	1
3	1
4	6
4	6
4	8
4	7
4	8
4	3
4	15
4	4
4	2
4	0