IT3708 Assignment 2

Marius Aarsnes

March 2019

1 MOEA

For this assignment I have chosen to use the NSGA-II, non-dominated sorting genetic algorithm. The benefits of this algorithm is that is fast and simple. Another algorithm I could have used is SPEA, Strength Pareto Evolutionary Algorithm. This algorithm is somewhat more complex than NSGA-II.

NSGA-II implements elitism by using a non-dominating sorting approach, sorting each individual in the population by domination rank. The domination rank of an individual is decided by the number of other individuals with it is dominated by. This is trivial to compute, and the individuals with the best domination rank are always passed on to the next generation. To enforce diversity, NSGA-II uses crowding distance to select a portion of the next generation. Individuals with a large crowding distance are further away from their neighbours and will therefore diversify the population. The crowding distance is used when there is not enough room for a whole set of individuals with the same rank x.

SPEA implements elitism by keeping an archive of only non-dominated-solution. New individuals are compared to the whole population and given. If they are not dominated and the archive is not full,the individual is added to the archive. The archive is used for offspring creation and only contains non-dominated solutions. SPEA favours diversity using the following method: when the number of non-dominated solutions is larger than the archive size, the less crowded solutions are kept in favour of more crowded solutions. This way, the Pareto front is kept diversified.

The main benefits of the NSGA-II is that it implements good and fast elitism. SPEA, on the other hand, is better when it comes to diversity among the non-dominated solutions. However, this comes at the cost of slower convergence. They are both relatively simple to calculate.

I believe NSGA-II is should be the preferred choice for this assignment. Due to the time constraints during the demo, in addition to limited memory and computing power required to get a decent solution the faster MOEA may be preferable. SPEA would have been beneficial for some of the more difficult images. MOEA may have the disadvantage of being more prone to converge to a local optimum, while SPEA keeps the Pareto front diverse and better avoids this.

2 Representation

The chromosome representation I have used for this assignment is a `1D int array`. The value at index `i` is the `id` of a neighbouring pixel, which the pixel at index `i` is pointing at. Each position in the array contains an integer in the interval `[0,numberOfPixels]` or `NULL`. The representation with a corresponding example is shown in Listing 1 and Table 1.

Listing 1: 1D Array Representation

```
Integer[] genotype = new Integer[this.pixelCount]
```

index	0	1	2	3	4	5	6	7	8	9
value	1	3	5	-	3	-	2	-	7	8

Table 1: Example of a chromosome

By viewing these paths as a sort of primitive disjoint set, each segment can be generated by iterating through the array and creating new segments for each `NUll` reached. Every index on the path to a `NULL` is added to the corresponding segment. The benefits of this representation are that it is simple, fast to convert to and small changes to the array could have a dramatic effect on the segments without sacrificing much processing power. Combining two different arrays or simply changing some values in an array can produce completely new segments of the image.

The downside to this representation is the difficulties with recombining after a crossover. This is due to the fact that when changing the array you do not know what pixels constitute the root nodes in the segments. To find them, the whole array needs to be traversed. Also, when doing the crossover, cleanup is required; there may be loops in the array. To fix these, one can either change which neighbour the pixels in the loop are pointing at, or a new segment can be created from the loop.

Another possible representation could be a 3D Boolean array. The array would represent the image as a matrix with a list of `true` or `false` values. At position `[row,col]` you store an array containing Boolean edge values stating whether the pixel at `[]row,col]` is connected to its neighbours. This representation is probably not as fast in the crossover. However, it is simpler and fast in the recombining phase as all the edges of every pixel are known. The only thing that needs clean-up is the consistency between pixels.

During the initialisation phase, I run a simple pre-segmentation of the image splitting it into "super-pixels". Each super-pixel consists of a set of original pixels based on the Euclidean distance in the xy-space and colour. This way I reduce the number of pixels the MOEA is working with during the whole evolution. Therefore, the cost of traversing all the pixels is greatly reduced. Due to the speedup from the initial segmentation, the downsides with the selected representation are combated. This, combined with its simplicity described above constitutes why I believe my representation is a fitting choice.

3 Findings

The parameters used for weighed sum is listed in **??**.

Minimum number of segments	maximum number of segments	mutation rate	generations
1	50	0.3	100

Table 2: Parameters used when testing

Overall deviation | Connectivity Measure | Edge value	Number of segments	Overall deviation | connectivity measure | edge value of solution 1	Average PRI score
0.3 0.3 0.3	6	17.5 48.3 42.4	83.66%
0.6 0.3 0.0	4	21.4 28.7 24.0	80.19%
0.0 0.6 0.3	1	47.3 00.0 00.0	43.06%
0.3 0.0 0.6	50	10.5 207.1 94.8	60.70%

Table 3: Weights used when testing

The measures `overall deviation` and `edge value` both favour a large number of segments. `Overall deviation` is calculated by the colour difference within a segment. It is the euclidean distance from the average colour of the segment to each of the pixels in the segment. When this value is `0.0` the whole segment has the same colour, and this is optimal. `Edge value` describes the contrast between the segment borders, and is a value for maximisation. `Edge value` favours many segments because if you have multiple segments with high contrast between them, this will be better than having a few large segments with high contrast between them.

The `connectivity measure`, on the other hand, favours a small number of segments. The `connectivity measure` is larger the more neighbouring segments are in the different segments and is a value which we are trying to minimise.

As the data in Table 3 and images below show, there is quite some difference when it comes to the number of segments for the solutions when weighing the parameters differently. When all parameters are weighed equally the algorithm finds a decent segmentation of the image, splitting it into clearly visible parts. The same goes for Figure 2, where the edge value is weighed 0 while the overall deviation is weighed more. When both overall deviation (or edge value) and connectivity measure is present, the two objectives are working against each other stabilising at a given number of segments, somewhere between the min. and max. allowed number of segments.

When one of the opposing factors is not present, the number of segments converge to either the minimum or the maximum, depending on which objective is not present/or weighed more. This can be seen in Figure 3 and Figure 4. When `connectivityMeasureWeight = 0`, the number of segments converges to the maximum number of allowed segments. This is because the two other objectives both favour many segments. We also see that in this case, the `connectivity measure` is much larger (and therefore worse) than when `connectivityMeasureWeight > 0`. When `overallDeviationWeight = 0`, the number of segments converges to the minimum allowed number of segments which is 1. This is due to the `connectivity measure` out-weighing the `edge value` and therefore few segments is favoured. Note that even though `edgeValueWeight = 0` for Figure 2, the edge value for the solution is not `0`. The same would have been the case for `overall deviation` if values of `overallDeviationWeight` and `edgeValueWeight` were flipped. This is due to the fact that these two are so strongly connected.

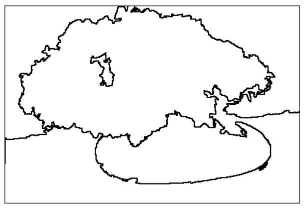
Figure 1: 0.3 0.3 0.3

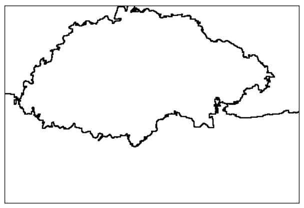
Figure 2: 0.6 0.3 0.0

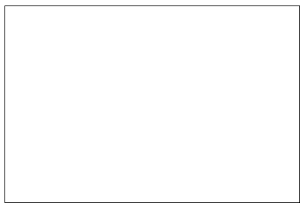
Figure 3: 0.0 0.6 0.3

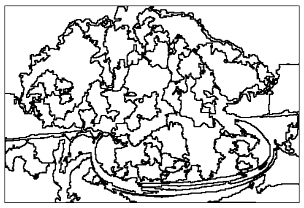
Figure 4: 0.3 0.0 0.6