ITNPBD8 – Evolutionary & Heuristic Optimisation

Course Assignment

David Haveron – Student Number 2527317

Contents

Problem introduction

The solutions outlined in this report, detail the process followed in the development of a colour sorting algorithm, which would enable users to sort through their collection of photos in a visually pleasing way. The illustration below illustrates a set of unordered colours and a set which have been have been sorted (optimised) using a 'direct' RGB Hue sort function in Python (1000 colours, as an example).

Figure 1 - 'Unsorted' order above & 'Sorted' order below

There are many ways to define how this is best achieved, however the scope of this report aims to demonstrate the application of 3 optimisation algorithms applied to minimise the distance between any two given colours from a file containing Red Green Blue (RGB) colour co-ordinates. The three algorithms were then applied to files containing 10, 100 and 1000 colours and their respective performances evaluated and discussed. The source code was written and prototyped in Python and has been included in the appendix.

The algorithms applied to this problem:

1. **First-improvement local search**
   This strategy iterates the process of generating a random solution, evaluating any candidate neighbourhood solutions (mutated solutions) and only accepting the neighbourhood solution which is an improvement from the current working solution – this continues until a predefined stop criterion is met.

2. **Iterated local search**
   An extension on the hill climber, this algorithm includes an additional loop to conduct the hill climbing algorithm multiple times and includes a perturbation operation for effectively escaping local minima and adding a 'momentum' to for finding the global minimum. This too includes a predefined stop criterion.

3. **Evolutionary algorithm**
   A population of individuals is generated and a steady state mutation/recombination operation applied to replace weak solutions with improved solutions, until a stop criterion is met.

The optimisation objective function for this problem space is initially defined as the minimisation of the Euclidean distance between two consecutive colours within the colour file. As the data file representing the photo order, consists of many colours represented by their RGB co-ordinates, we define the fitness function or more appropriately the distance function of the colour set as the sum of the individual distances between consecutive colours in entire dataset. The Euclidean distance function has been defined below:

$$\textbf{Distance}\ (\ R_iG_iB_i\ ,\ R_{i+1}G_{i+1}B_{i+1}\ ) = \sum \sqrt{(R_{i+1}-R_i)^2+(G_{i+1}-G_i)^2+(B_{i+1}-B_i)^2}$$

Optimisation functions are usually subject to functional constraints. For the problem outlined so far in this report it should be noted that a column vector is mapped to the initial RGB colours presented in the data file of length 10, 100 or 1000. It follows the solutions are constrained to be permutations

of the original column vector – that is, a finite set without repeating integers (hard constraint). Additional constraints discussed in this report are the computational limitations inherent in the system which will support this product (hard constraints) and, the performance expectations for users of this product (soft constraint) – these all are to be considered with the design/choice of an optimisation algorithm.

Rationale for move operators and termination criteria

This report outlines the two search/variation operators which were considered, namely the *colour_swop* and the *colour_invert* operators. Given an example set of 10 colours, an initial solution could be defined by assigning an index to each of the 10 colours. The resulting solution could be a list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. Passing this solution through the *colour_swop* operator would result in two index positions being interchanged, or [0, 1, **5**, 3, 4, **2**, 6, 7, 8, 9]. The *colour_invert* operator takes a slice of random length, at a random position and inverts the slice with respect to order. Concretely an initial example solution of [0, 1, 2, 3, 4, 5, **6**, **7**, **8**, 9] may become [0, 1, **8**, **7**, **6**, 2, 3, 4, 5, 9] or [0, 1, 2, 3, 4, 5, 9, **8**, **7**, **6**]. The two operators were considered for both the *Hill-Climber* and *Iterated Local Search* algorithms, their performance summarised in the table below (over 1000 iterations):

		Euclidian Distance	**Time (s)**
10 colours	colour_swop	4.46	9.88
	colour_invert	4.15	7.54
100 colours	colour_swop	58.71	52.88
	colour_invert	38.43	37.49
1000 colours	colour_swop	643.52	461.09
	colour_invert	580.41	313.72
Scaled Average (assuming 1000 colours)	colour_swop	558.87	659.30
	colour_invert	459.90	480.87

Although this performance table reflects the performance of a single (versus multiple) run(s), it can be proven the *colour_invert* operator proves a more effective mutation method for escaping local minima (the proof is outside of the scope of this report).

The termination criteria prove a more difficult to quantify and define. Although in an ideal situation one would leave these algorithms running to exhaustively explore the search space, this would not be computationally or practically feasible. Because of these computational and practical constraints, a heuristic (approximate solution) approach needs to be taken where is feasible candidate solution given the system constraints, is chosen. Modern day software and products often fail to be (generally) adopted by consumers if there is an unusually large latency while waiting for a result, so the design of the colour sorting product needs to incorporate these considerations when choosing an appropriate system. Software systems now days are expected to return results within milliseconds.

For the choice of stop criterion, the algorithms were left to run through 5000 iterations of Hill-Climber 'sorting' to understand the 'approximate convergence' as the example plots (*Figures 3,4,5*) illustrate below. It was decided that should a 'satisfactory sort of colours' be achieved and hence approximate/heuristic solution found, the algorithm loops break. Given the computational constraints on the system (my computer), this approach returns a satisfactory/approximate solution

without taking a displeasing amount of time to complete. An approximate stop criterion for the 10, 100 and 1000 colours was coded to be 5.62, 43.12 and 682.32 based on an increase of the mean values 4.62, 33.12 and 582.32 which were found. This approach is bespoke and appropriate for choosing a generalized stop criterion, however the same methodology could be followed with the Iterated Local Search and Evolutionary Algorithms to find a satisfactory stop criterion which balances the opposing 'effectiveness of sort' and 'computation` (latency).

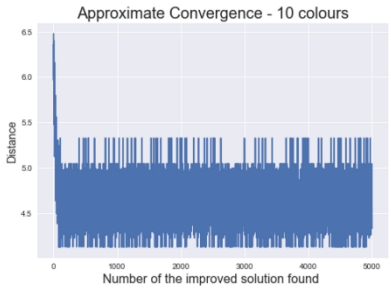


Figure 2 - Convergence, 10 Colours

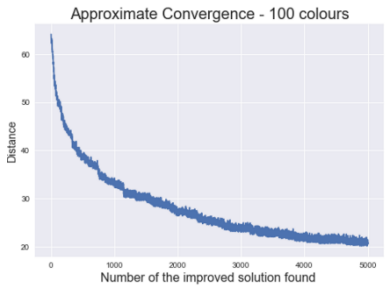


Figure 3 - Convergence, 100 Colours

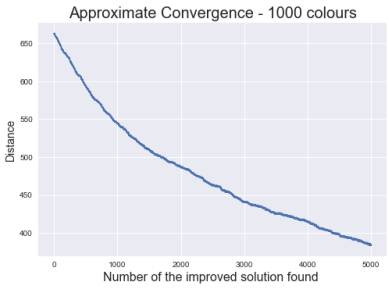


Figure 4 - Evolutionary Algorithm

Results

The three algorithms discussed all aim to yield a heuristic solution (an optimised order of RGB coordinates) which represents a minimised accumulative Euclidean distance between sequential RGB coordinates in the file, when compared to the initial solution. The objective of running each of these algorithms once, is to produce a single heuristic solution to the objective function. In the case of the Hill-climber and Iterated Local search, a best solution is held in record as the final output solution. In the case of the Evolutionary algorithm, a parent with the highest fitness, or lowest accumulative distance rather is recorded as the final output solution. The three algorithms were run 30 times each, for a separate set of 10, 100 and 1000 colours. The mean and standard deviation were calculated across the results from the 30 runs to explore algorithm robustness, and, the respective algorithm performance for the three approaches have been summarised below in this section. Note the data produced from the 30 runs, for each of the three algorithms has been included in the appendix.

10 Colours

	Mean	Std. Deviation	Best (lowest) Distance Found	Average Time/run (s)
Hill-climber	4.52	0.09	4.30	0.40
Iterated local search	4.45	0.10	4.15	0.78
Evolutionary algorithm	4.49	0.09	4.30	18.23

Random Solution

Hill-climber

Iterated Local Search

Evolutionary Algorithm

100 Colours

	Mean	Std. Deviation	Best (lowest) Distance Found	Average Time/run (s)
Hill-climber	53.45	2.18	48.98	4.3
Iterated local search	39.32	1.88	35.08	55.4
Evolutionary algorithm	57.56	1.12	55.47	6.07

Random Solution

Hill-climber

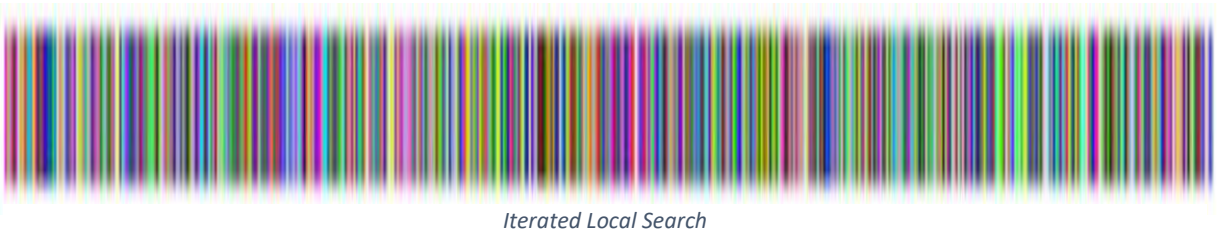
Iterated Local Search

Evolutionary Algorithm

1000 Colours

Random solution

Hill-climber



Iterated Local Search

Evolutionary Algorithm

	Mean	**Std. Deviation**	**Best (lowest) Distance Found**	**Average Time/run (s)**
Hill-climber	658.50	7.48	642.92	0.97
Iterated local search	659.52	8.15	646.48	2.23
Evolutionary algorithm	644.51	4.05	637.72	7.03

Algorithm comparison

The plot framework has been included in the code, and would be a beneficial tool to visualise and compare algorithm performance.

Discussion

The approach outlined in this report included 30 runs for each of the Hill-climber, Iterated-Local-Search and Evolutionary algorithms to test algorithm robustness. As the Hill-climber and Iterated-Local-Search algorithms explore a large solution search space, there is no guarantee the algorithms will find an optimal solution for a given run, with the practical and computational constraints we impose on the system. Similarly, the Evolutionary-algorithm may not converge to an optimal solution space given the same constraints for a given run.

During the evaluation of these algorithms, it was found the Hill-Climber algorithm fell into local minima and failed to meet the stop criterion given the number of iterations imposed on the system. Like the Hill-climber, the evolutionary algorithm too is inappropriate for this application, as the population mean increases (improvements to the worst solutions), however as no improvement is made to the parents, the parents limit the choice of best solution (this is left to chance of a superior child being created).

Of the three algorithms evaluated in this report, the Iterated Local Search Algorithm appeared to be most suited for finding satisfactory solutions and escaping local minima and is suggested as an appropriate heuristic approach to be used in the development of the colour sorting product.

Following this, the choice of algorithm for this colour sorting product has been based on mean performance across the 30 runs, opposed to a single run for which the outcome may indeed be non-representative generalized algorithms performance. Additionally, there is an assumption the algorithm performance and hence product (sorting) performance is dependent on the computational infrastructure supporting it, so an approach is suggested to apply the same methodology detailed in this report however re-evaluated using the final computational infrastructure on which it will be run commercially. This will give a more clear and accurate understanding of each of the algorithms performance and robustness for solving the given problem.