This report discusses the procedure used and the results obtained from three Computational Intelligence optimisation algorithms, namely, a Genetic Algorithm (GA), Differential Evolution (DE) and Particle Swarm Optimisation (PSO), used to solve George Stigler's diet problem from 1939.

Diet Problem Optimisation

WRCI411 Assignment 4

Jason Collier – 214008258 – 02/10/2017

Contents

Introduction

George Stigler, a Nobel Laureate, identified the Diet Problem in 1939. Stigler made a list of 77 common food items, along with their respective nutritional contents. His goal was to meet the average annual nutritional intake requirements for an adult male (see table 1) using the cheapest combination of food items possible. Stigler found that he could meet the average annual adult male dietary requirements for $39.93 per year.

Stigler's solution to the Diet Problem was optimised further in 1947 by nine clerks, working for over two weeks, using Dantzig's Simplex Algorithm. They found that the dietary requirements could be met for $39.69 per year.

This report discusses solutions obtained from three Computational Intelligence optimisation algorithms, namely, a Genetic Algorithm, a Differential Evolution algorithm and a Particle Swarm Optimisation algorithm, which are used to solve the Stigler's Diet Problem.

Item	**Amount**	**Unit**
Calories	1095	Kilo Calories
Protein	25550	grams
Calcium	292	grams
Iron	4380	mg
Vitamin A	1825	Kilo IU
Thiamine VB1	657	mg
Riboflavin VB2	985.5	mg
Niacin	6570	mg
Ascorbic Acid VC	27375	mg

Table 1: Minimum average nutritional needs for an adult male

Theory

Genetic Algorithm

Genetic Algorithms are loosely based on the evolution of mankind. They consist of a population of individuals which is improved over generations through a “survival of the fittest” approach. Selection methods such as tournament-based selection add an element of randomness to the algorithm. Crossover rates and mutation rates add diversity to the population to ensure the best solution is reached. The effects of manipulating these parameters is discussed in the Discussion section. The Genetic Algorithm is summarised below:

```
Initialise a population n random individuals
While stopping conditions NOT met
        Evaluate fitness of each individual;
        Select individuals for reproduction;
        Create offspring (crossovers and mutations);
        Select new population C(t + 1);
End
```

Differential Evolution

Differential Evolution is a stochastic, population-based search strategy, like Genetic Algorithms, but it differs in the sense that the distance and direction information from the current population is used to guide the search process. The scaling factor is used to control the step sizes of the differential variations which influences the diversity and converging time. The recombination probability influences the diversity of the population directly. The effects of manipulating these parameters is discussed in the Discussion section. The Differential Evolution algorithm is summarised below:

```
Initialise a population n random individuals
While stopping conditions not met
        //Produce u
        For i = 1 to n
                Select randomly $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$;
                $\mathbf{U}_i = \mathbf{x}_1 + \beta (\mathbf{x}_2 - \mathbf{x}_3)$;
        end

        //Produce x'
        $x_{ij}(t + 1) = u_{ij}$ if $j \varepsilon J$; OR $x_{ij}(t + 1) = x_{ij}(t)$ otherwise;

        //Produce x
        For i = 1 to n
                Replace $\mathbf{x}_i$ with $\mathbf{x}_i'$ if $\mathbf{x}_i'$ better;
        End
end
```

Particle Swarm Optimisation

The Particle Swarm Optimisation algorithm is a population-based search algorithm based on the simulation of birds within a flock. Changes to particles in a swarm are influenced by changes in their neighbours. Particles consequently always stochastically return to previously successful regions in the search space on their path toward the best answer. The population size, the neighbourhood size and the inertia weight are all parameters which affect the Particle Swarm Optimisation which will be discussed in the Discussion section. The Particle Swarm Optimisation algorithm is summarised below:

```
Initialise a swarm n random particles
While stopping conditions not met
    For i = 1 to n
        //Set the personal best position
        If $f(\mathbf{x}_i) < f(\mathbf{y}i)$ then
            $\mathbf{y}_i = \mathbf{x}_i$;
        End

        //Set the neighborhood best position
        If $f(\mathbf{y}_i) < f(\hat{\mathbf{y}}_i)$ then
            $\hat{\mathbf{y}}_i = \mathbf{y}_i$
        end
    end

    For i = 1 to n
        $\mathbf{V}_{ij}(t + 1) = w * \mathbf{v}_{ij}(t) + c_1 * \mathbf{r}_{1j}(t) * (\mathbf{y}_{ij}(t) - \mathbf{x}_{ij}(t)) + c_2 * \mathbf{r}_{2j}(t) * (\hat{\mathbf{y}}_j(t) - \mathbf{x}_{ij}(t))$
        $\mathbf{X}_{ij}(t + 1) = \mathbf{x}_{ij}(t) + \mathbf{v}_{ij}(t + 1)$
    End
end
```

Method

Each of the three algorithms were first programmed individually until working algorithms were obtained. The Differential Evolution algorithm, which showed the best results initially, was used to optimise the fitness function using a trial and error method. This fitness function was then implemented in the Genetic Algorithm and the Particle Swarm Optimisation Algorithm. The respective parameters of all three algorithms were tweaked by trial and error until each of them found acceptable solutions.

All the algorithms use populations consisting of individuals of 77 genes each, one for each food type's mass in grams. For comparison purposes, each of the three algorithms use an initial population size of 200 individuals with each of every individual's 77 genes having an initial mass randomised between 0 and 10 dollars' worth of each food item.

The fitness function used by all three algorithms has three terms, namely, cost, the sum-squared-error between the required dietary requirements and the individual's nutritional values when the nutritional value of the individual is under the requirements, and a count of the number of the nine requirements which are met. The fitness method is shown below:

```
int c = 12;
int s = 2;
int u = 30;
int o = 10;
for (i = 0 to 77)
{
Cost += curMass * (dollars / gram)
//Calculate the total quantity of each nutritient for current individual
KiloCalories +=
Protein +=
etc
}
for (each nutrient)
{
        double val = curIndivNutInfo – minNutIntake;
        if (val < 0)
                SSE += (u * val) / minNutIntake)^2;
        else
                over++;
}
 fitness = (c * cost) + (s * SSE) – (o * over);
```

C, s and n are constants used to alter the importance of each term. The fitness function minimises cost of each individual, raises all nutritional values which are under those required and rewards individuals when the requirements are met.

Once the problem has been solved by each algorithm, that algorithm's parameters will be used as its original parameters. Each of the algorithms parameters will be manipulated and compared to that algorithm's original parameters. The solution found using the best set of parameters of each algorithm will be compared to one another.

Genetic Algorithm

Stopping Conditions	STOP when t is GREATER THAN 2500 iterations OR when population fitness has not changed for 100 iterations
Initialization	Mass of each food item initialized in the range [0, 10] dollars' worth
Values for population size investigated	50, 200, 500, 1000
Values for mutation rate investigated	0, 0.2, 0.4, dynamic (probabilities for uniform mutation)
Values for mutation magnitude investigated	random values in the range [0, α] where $\alpha = x_{max,j} - x_{ij}$ or $\alpha = x_{ij} - x_{min,j}$ (selected at random)
Values for selection method investigated	5%, 20%, 50%, 100% of total population (tournament selection)
Values for crossover rate investigated	0.25, 0.5, 0.85, 1.0 (probabilities for uniform crossover)

Table 2: Implementation details for the Genetic Algorithm

The original parameters for the Genetic Algorithm, to which tests will be compared, will include a population size of 200, a dynamic mutation rate (starting large and decreasing as the generations go by), a tournament selection size of 5% of the total population and a crossover rate of 0.85.

Differential Evolution

Stopping Conditions	STOP when t is GREATER THAN 30000 iterations OR when population fitness has not changed for 1000 iterations
Initialization	Mass of each food item initialized in the range [0, 10] dollars' worth
Values for population size investigated	50, 200, 500, 1000
Values for scale factor investigated	0.5, 0.75, 1.0, 5.0
Values for crossover rate investigated	0.3, 0.5, 0.7, 1.0 (probabilities for binomial crossover)

Table 3: Implementation details for the Differential Evolution algorithm

The original parameters for the Differential Evolution algorithm, to which tests will be compared, will include a population size of 200, a scale factor of 0.7 and a crossover rate of 0.3.

Particle Swarm Optimisation

Stopping Conditions	STOP when t is GREATER THAN 1000 iterations OR when population fitness has not changed for 100 iterations
Initialization	Mass of each food item initialized in the range [0, 10] dollars' worth
Values for population size investigated	50, 200
Values for inertia weight investigated	0.25, 0.877, 1.0
Values for neighbourhood size investigated	25%, 50%, 75%, 100% of total population

Table 4: Implementation details for the Particle Swarm Optimisation algorithm

The original parameters for the Differential Evolution algorithm, to which tests will be compared, will include a population size of 200, an inertia weight of 0.877 and a neighbourhood size of 75% of the total population.

Results

Genetic Algorithm

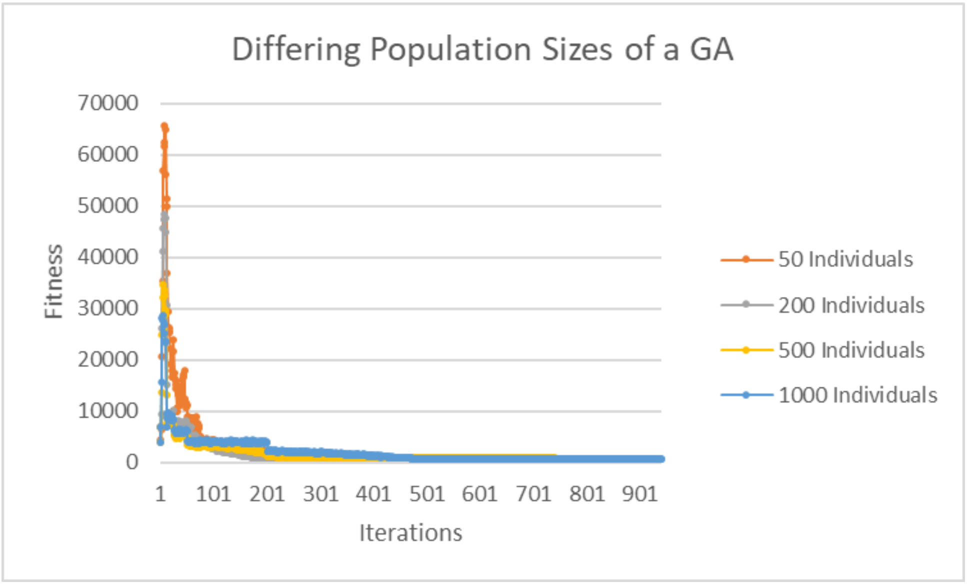

Figure 1: A graph comparing differing population sizes of a Genetic Algorithm

Population Size	Iterations	Fitness	Cost
50	329	1615	$76.11
200 (original)	355	866	$50.16
500	741	852	$49.63
1000	941	819	$48.32

Table 5: The results found by differing population sizes of a Genetic Algorithm

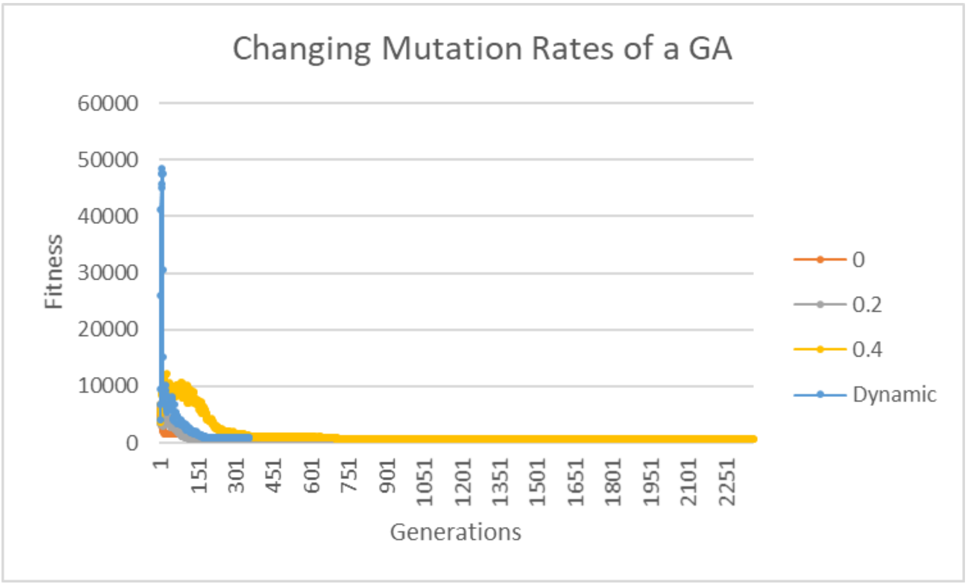

Figure 2: A graph comparing changing mutation rates of a Genetic Algorithm

Mutation Rate	Iterations	Fitness	Cost
0.0	131	1617	$92.49
0.2	1285	758	$47.9
0.4	2362	776	$48.5
Dynamic (original)	355	866	$50.16

Table 6: The results found by changing the mutation rate of a Genetic Algorithm

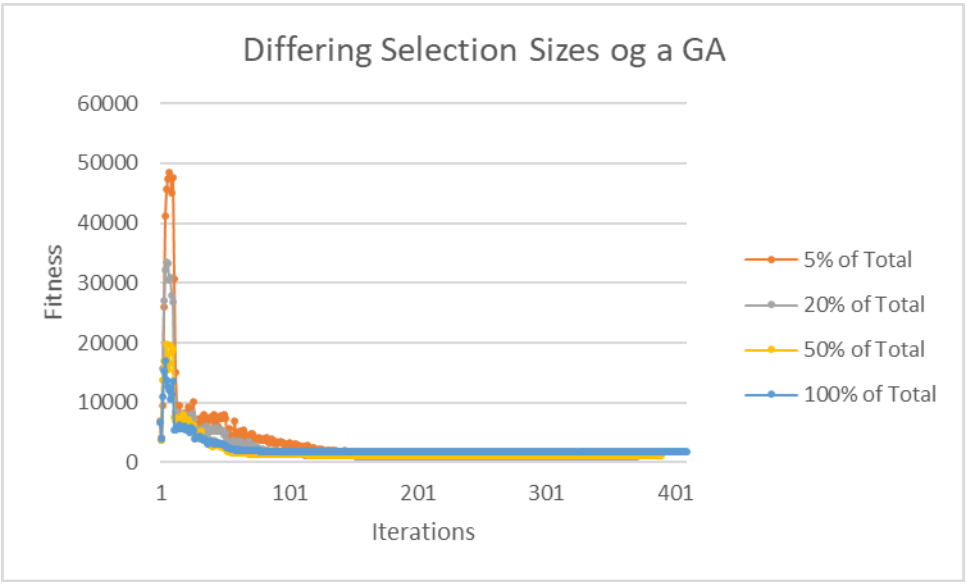

Figure 3: A graph comparing differing selection sizes of a Genetic Algorithm

Selection Size	Iterations	Fitness	Cost
5% of total (original)	355	866	$50.16
20% of total	371	995	$56.68
50% of total	390	1259	$61.55
100% of total	410	1771	$86.13

Table 7: The results found by differing the selection size of a Genetic Algorithm

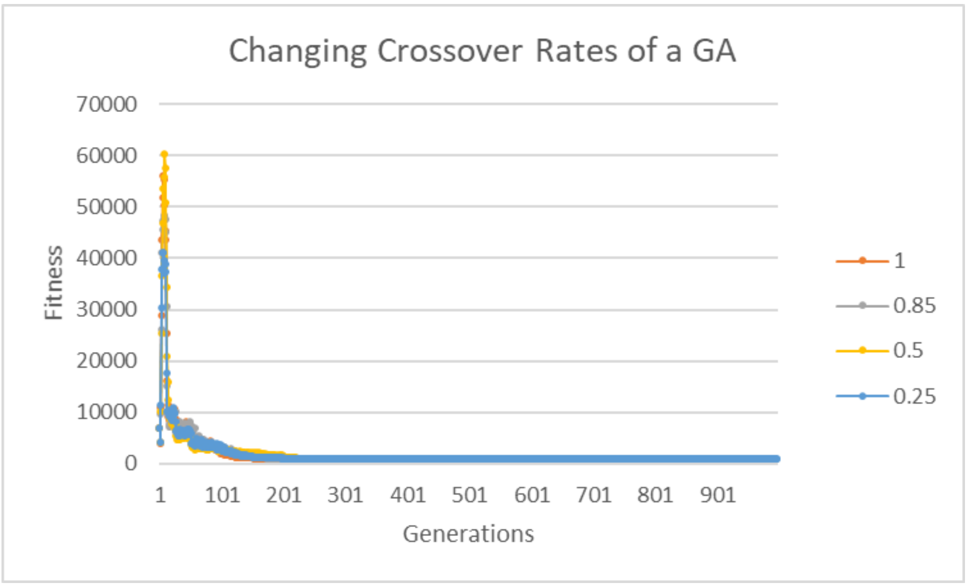

Figure 4: A graph comparing changing crossover rates of a Genetic Algorithm

Crossover Rate	Iterations	Fitness	Cost
0.25	993	964	$57.05
0.5	954	916	$54.66
0.85 (original)	355	866	$50.16
1.0	350	864	$50.09

Table 8: The results found by changing the crossover rate of a Genetic Algorithm

Iteration	Time (min)	Cost ($)	Kilo Calories	Protein (g)	Calcium (g)	Iron (mg)	Vitamin A (Kilo IU)	Thiamine VB1 (mg)	Riboflavin VB2 (mg)	Niacin (mg)	Ascorbic Acid VC (mg)
1110	55	45	1057	37407	292	13400	3239	1087	986	6991	27375
Required:			1095	25550	292	4380	1825	657	985.5	6570	27375

Table 9: The best results found by the GA using a population size of 1000, a mutation rate of 0.2, a selection size of 4 % and a crossover rate of 0.9

Differential Evolution

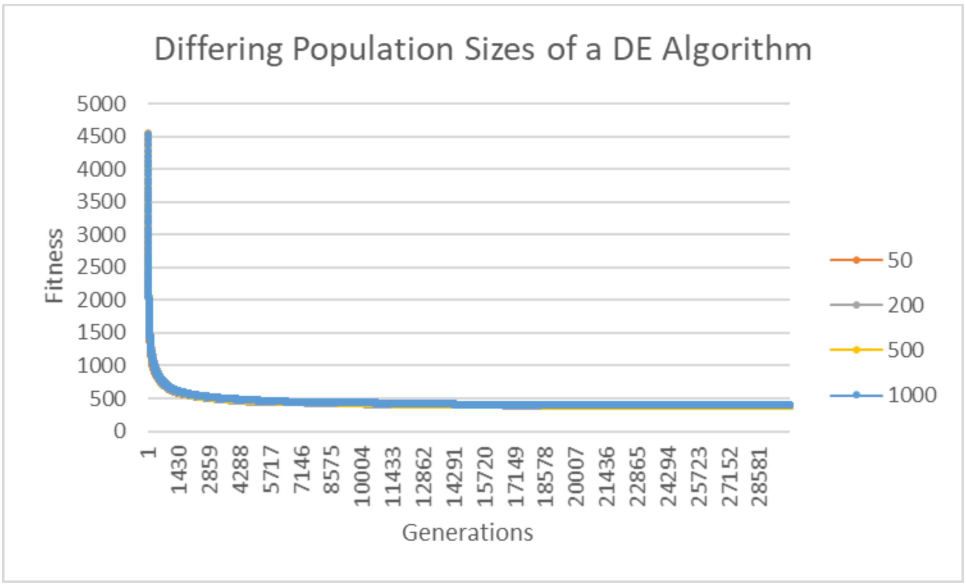

Figure 5: A graph comparing differing population sizes of a Differential Evolution algorithm

Population Size	Iterations	Fitness	Cost
50	30000	391.09	$40.06
200 (original)	30000	391.69	$40.06
500	30000	391.11	$39.93
1000	30000	392.81	$39.93

Table 10: The results found by differing population sizes of a Differential Evolution algorithm

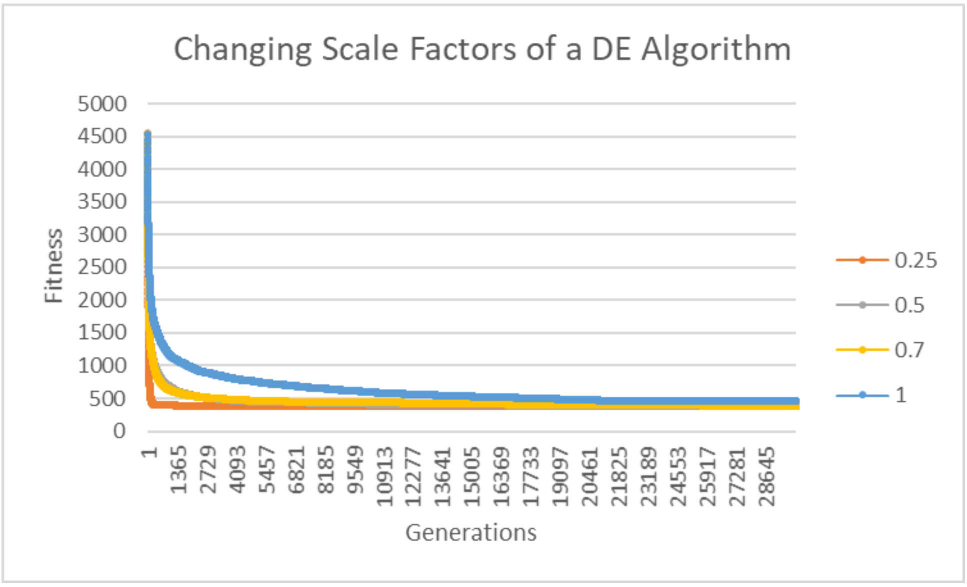

Figure 6: A graph comparing changing scale factors of a Differential Evolution algorithm

Scale Factor	Iterations	Fitness	Cost
0.25	30000	387.58	$39.80
0.5	30000	390.97	$39.90
0.7 (Original)	30000	391.69	$40.06
1.0	30000	464.62	$40.91

Table 11: The results found by changing the scale factor of a Differential Evolution algorithm

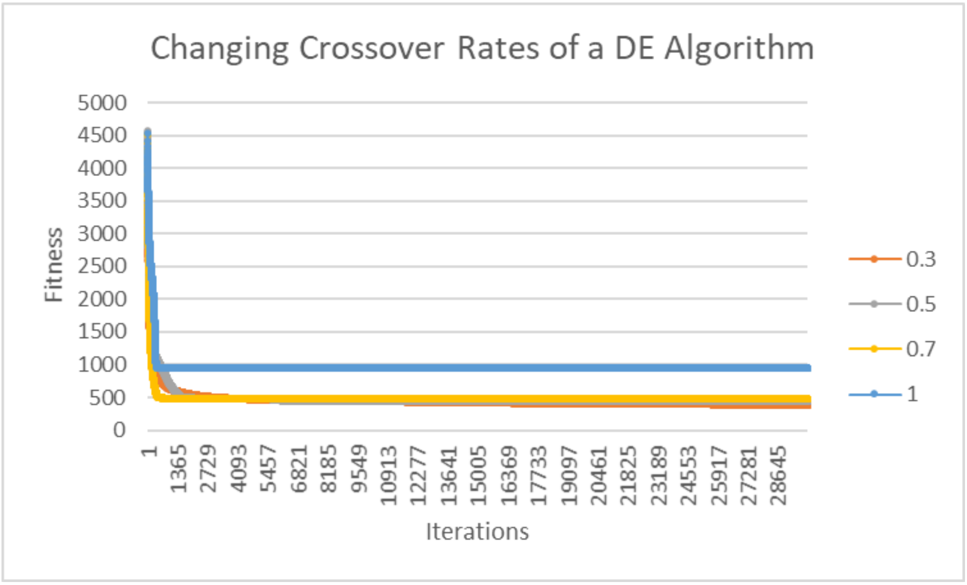

Figure 7: A graph comparing changing crossover rates of a Differential Evolution algorithm

Crossover Rate	Iterations	Fitness	Cost
0.3 (Original)	30000	391.69	$40.60
0.5	30000	434.18	$42.00
0.7	30000	489.56	$44.05
1.0	30000	946.58	$78.77

Table 12: The results found by changing the crossover rate of a Differential Evolution algorithm

Iter-ations	Time (min)	Cost ($)	Kilo Calories	Protein (g)	Calcium (g)	Iron (mg)	Vitamin A (Kilo IU)	Thiamine VB1 (mg)	Riboflavin VB2 (mg)	Niacin (mg)	Ascorbic Acid VC (mg)
1110	5	39.66	1095	53806	292	22070	1825	1504	986	9970	27375
Required:			1095	25550	292	4380	1825	657	985.5	6570	27375

Table 13: The best results found by the DE algorithm using a population size of 200, a scale factor of 0.25 and a crossover rate of 0.3

Particle Swarm Optimisation

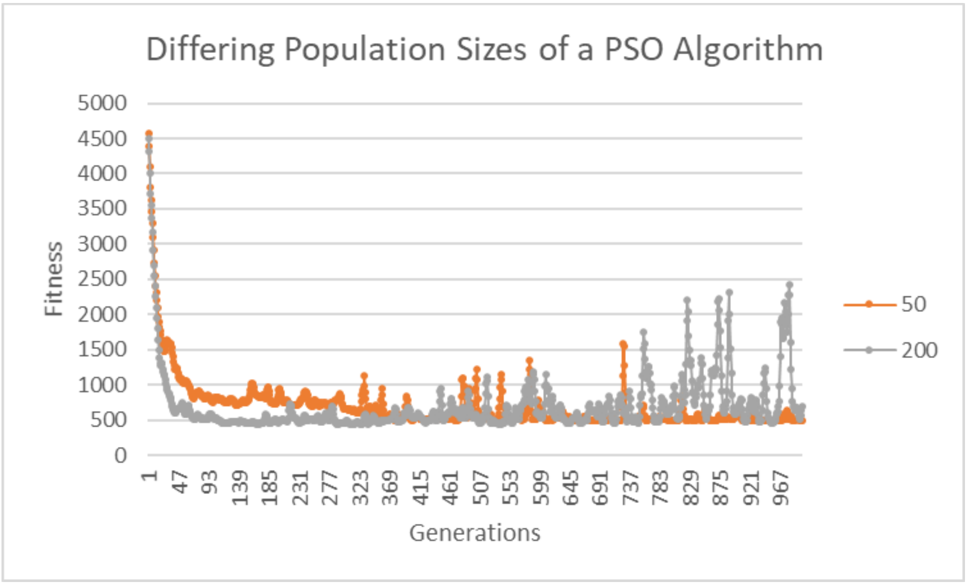

Figure 8: A graph comparing differing population sizes of a Particle Swarm Optimisation algorithm

Population Size	Iterations	Fitness	Cost
50	1000	505.18	$46.24
200 (original)	1000	630.41	$41.83

Table 14: The results found by differing population sizes of a Particle Swarm Optimisation algorithm

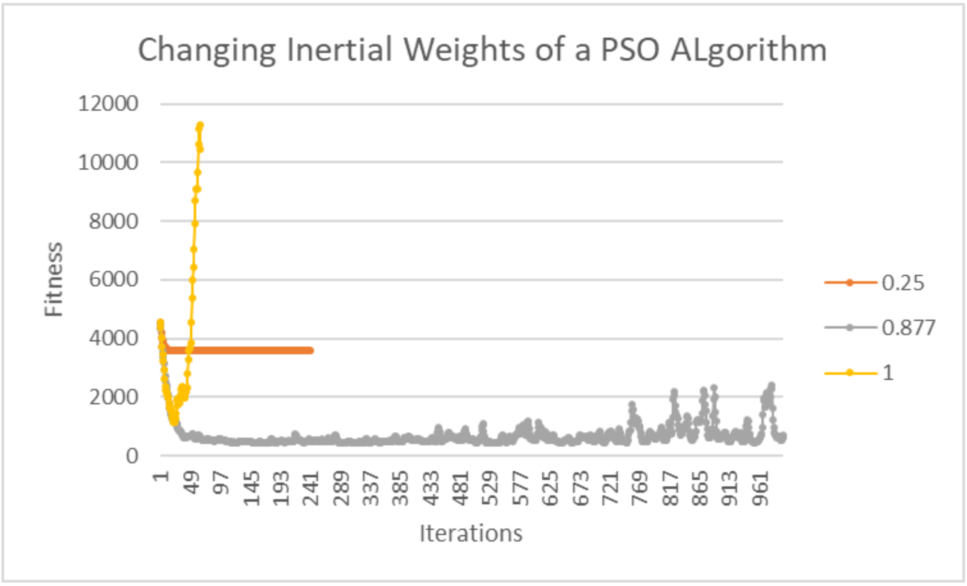

Figure 9: A graph comparing changing inertial weights of a Particle Swarm Optimisation algorithm

Inertial Weight	Iterations	Fitness	Cost
0.25	241	3577	$40.06
0.877 (original)	1000	630.41	$40.06
1.0	65	10439	NA

Table 15: The results found by changing the inertial weight of a Particle Swarm Optimisation algorithm

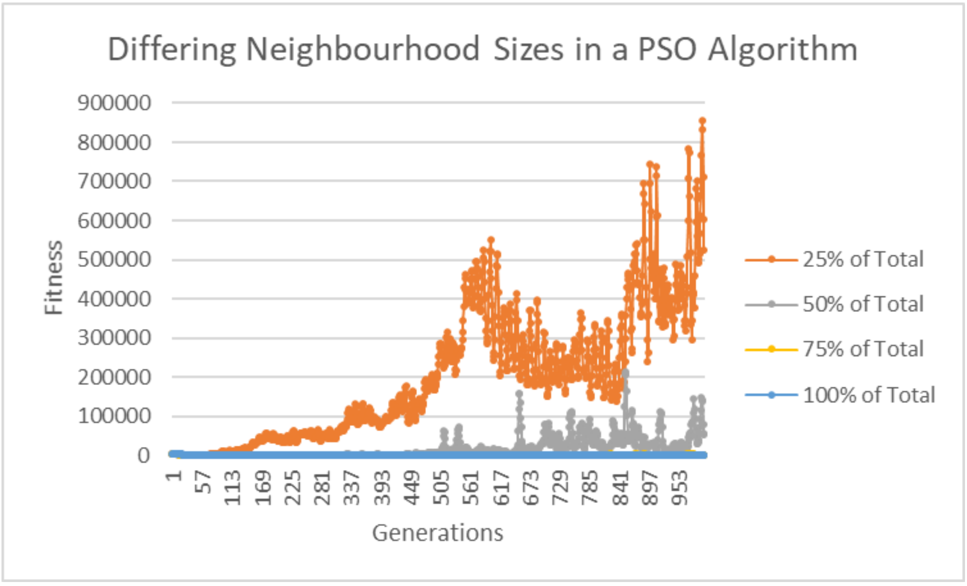

Figure 10: A graph comparing differing neighbourhood sizes of a Particle Swarm Optimisation algorithm

Neighbourhood Size	Iterations	Fitness	Cost
25%	1000	Extreme Fluctuations	$44.29
50%	1000	Fluctuations	$44.17
75% (Original)	1000	630.41	$41.83
100%	1000	446.18	$42.3

Table 16: The results found by differing neighbourhood sizes of a Particle Swarm Optimisation algorithm

Iter-ation	Time (min)	Cost ($)	Kilo Calories	Protein (g)	Calcium (g)	Iron (mg)	Vitamin A (Kilo IU)	Thiamine VB1 (mg)	Riboflavin VB2 (mg)	Niacin (mg)	Ascorbic Acid VC (mg)
1000	17	42	1013	53080	292	23425	9126	1421	994	9037	27375
Required:			1095	25550	292	4380	1825	657	985.5	6570	27375

Table 17: The best results found by the PSO algorithm using a population size of 200, an inertial weight of 0.877 and neighbourhood size of 150

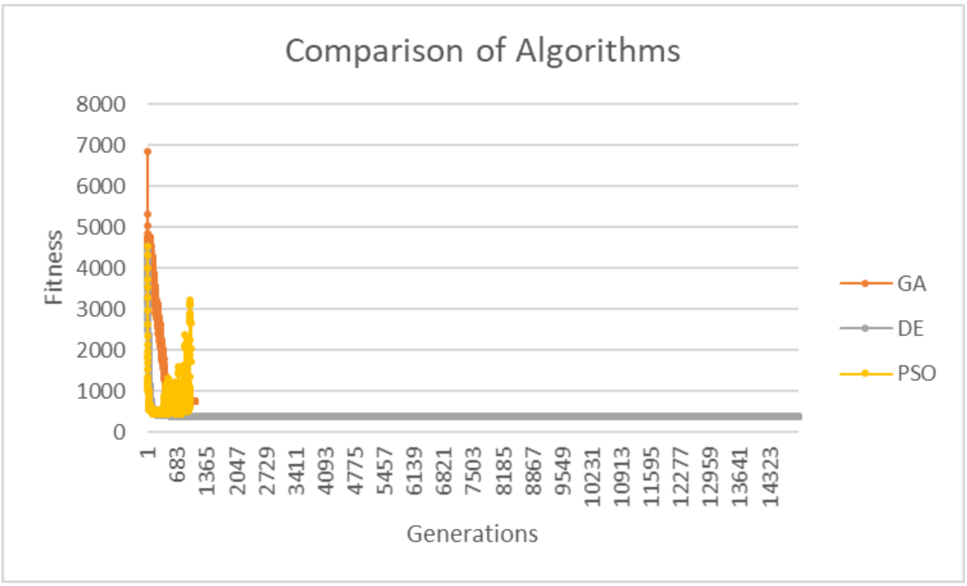

Figure 11: A comparison between the best solutions found by each algorithm

Discussion

Genetic Algorithm

The final settings chosen included a population size of 1000 individuals, a tournament selection size of 40 individuals, a mutation rate of 0.2, a crossover rate of 0.9 and an initialization range of [0, 10] dollars' worth of mass for each food item. These settings were chosen, based on the results depicted in figures 1 - 4 and tabulated in tables 5 - 8, to maintain population diversity while keeping efficiency in mind due to time constraints. These are the settings that resulted in the optimum masses of each food item and which resulted in the solution shown in table 9.

The algorithm stops when the population fitness does not change for 100 consecutive generations or when a maximum of 2500 generations is reached to prevent infinite loops from occurring. The results of different settings, using these stopping conditions, are graphed in figures 1 – 4 and are discussed below.

A small population of 50 was used in order to quickly verify whether the code was working or not. Once verified, larger population sizes of 200, 500 and 1000 were used. Larger populations than these were not considered as they slowed the algorithm down unnecessarily, so a population size of 1000 individuals was selected to maintain diversity and efficiency (see figure 1 and table 5).

Various percentages of this population, used in tournament selection to select parents, were investigated, namely 5%, 20%, 50% and 100%. The selective pressures are too high for selection groups of 20%, 50% and 100% of the total population, and the takeover time is too long for a selection groups smaller than 4% (see figure 2 and table 6). selection groups smaller than 4% also cause too many fluctuations in the solution over the generations. So, a selection group of 4% of the total population was used.

The new population is selected by taking the best individual of the previous population and filling up the rest of the population with offspring.

Mutation rates of 0, 0.2, 0.4 and a dynamic rate were investigated (see figure 3 and table 7). Higher mutation rates were not investigated as they made the converging time unnecessarily long and caused unnecessary fluctuations in the solution. A mutation rate of 0 was not chosen because it does not add diversity to the population and a mutation rate of 0.4 was not chosen because it added too much time to the algorithm process. The dynamic mutation rate was not chosen because it started off too high and distorted the good solutions. A mutation rate of 0.2 was decided on as it is big enough to add diversity and small enough not to distort the good solutions.

Varying the crossover rate does not have much of an effect on the algorithm (see figure 4 and table 8), but a crossover rate of 0.9 was used over the other rates tested in order to add an element of randomness and lower the selective pressure slightly, while not slowing the algorithm too much.

Differential Evolution

The final settings chosen included a population size of 200, scale factor of 0.25 and a crossover rate of 0.3. These parameters were chosen in order to ensure that the full search space was explored, to prevent premature convergence while still maintaining efficiency. Figures 5 – 7 and tables 10 – 12 were used to determine the optimal parameters. The resulting solution can be seen in table 13.

A population size of 200 was used in order to maintain a balance between the number of differential vectors available and the computational complexity. Smaller populations did not allow sufficient exploration and bigger populations were too taxing time-wise (see figure 5 and table 10).

The smaller scaling factor values cause step sizes to be smaller and thus causes the convergence time of the algorithm to take longer. Bigger scaling factor values facilitate exploration, but have been known to overshoot good optima. A value of 0.25 was decided on because it is small enough to allow differentials to explore tight valleys, but big enough to maintain diversity (see figure 6 and table 11).

Crossover rates smaller than the chosen 0.3 do not offer enough diversity. Crossover rates bigger than 0.3 converge too quickly and do not allow for a robust search. A crossover rate of 0.3 facilitates good exploration (see figure 7 and table 12).

Particle Swarm Optimisation

The best parameters found for the PSO algorithm included a population size of 200, a neighbourhood size of 150 and an inertial weight of 0.877. The graphs in figures 8 – 10, along with tables 14 – 16 were used to determine these settings. These parameters resulted in an optimal answer which can be seen in table 17.

A small population of 50 was used in order to quickly verify whether the code was working or not. Once verified, larger population size of 200 was used. Larger populations than these were not considered as they slowed the algorithm down unnecessarily, so a population size of 200 individuals was selected to maintain diversity and efficiency (see figure 8 and table 14).

An inertia weight is used to control the exploration and exploitation abilities of the swarm. The inertia weight controls the momentum of the particle by weighing the contribution of the previous particle. Based on trial and error, the best inertial weight was found to be 0.877 (see figure 9 and table 15).

The global best algorithm was first implemented, but it was noticed that it often converged to local minima instead of global minima. A Local Best algorithm, which resulted in better solutions at the expense of an extended convergence time, was then implemented in order to maintain diversity and aid exploration. A neighbourhood size of 75% of the total population was found to be best through trial and error (see figure 10 and table 16).

Conclusion

The Genetic Algorithm's best solution to Stigler's Diet Problem is shown in table 9. The algorithm did not optimise the diet problem as well as Stigler did. This Genetic Algorithm found that the lowest price possible to satisfy the average adult male's annual dietary requirements is $45.00 which is not as good as Stigler's solution of $39.93. This is due to the fact that the algorithm found the local best instead of the global best as diversity was not maintained throughout the training process.

The Differential Evolution algorithm provides a solution of $39.66 to Stigler's Diet Problem as can be seen in table 13. This is an exceptionally good solution.

The Particle Swarm Optimisation algorithm solved Stigler's Diet Problem with $42. This is not the best solution, but it is acceptable.

Based on the graph in figure 11, which compares the algorithms, one can clearly see that the Differential Evolution algorithm performs best. It converges faster than the other two algorithms and converges to a better solution than that of the other two algorithms. It also finds the optimal solution in 5 minutes which is notably faster than the Particle Swarm Optimisation algorithm which converges in 15 minutes and the Genetic Algorithm which converges in 55 minutes! The Particle Swarm Optimisation algorithm converges to a decent solution relatively quickly, but then begins to fluctuate and distort good solutions. The Genetic Algorithm learns slowly and converges to a bad answer.