A Variation on a Memetic Algorithm for boiler Scheduling

Sally Evans, Dr Ian Fletcher
Sunderland University, Sunderland, SR1 3SE, England

Abstract, This paper investigates the optimisation of pressurised steam production for a power plant, via a number of interconnected boilers of varying efficiency characteristics. The aim of the scheme is to provide the required amount of steam whilst allowing sufficient spare capacity to compensate for boiler failure.

The proposed solution uses evolutionary techniques based on genetic and memetic algorithms. The results were compared with those from a solver based on linear and quadratic programming. For operational purposes, the process has to be optimised at least every fifteen minutes.

1. Introduction

The initial problem was posed in an industrial setting, based on a real power station as part of a larger problem. The company, MDC, involved is a software company selling optimisation software to the chemical and power production industries.

The research was an experiment to explore the use of evolutionary algorithms and compare them to another method, namely the solver used in Excel, which uses linear programming and quadratic programming.

The aim was to calculate the minimum cost to produce a given amount of steam from a series of boilers, each boiler having a different efficiency function, modelled here by a quadratic function.

The constraints on the problem are:

- There should be spare capacity should one of the boilers fail.
- Each boiler has finite capacity.

The cost of fuel needs to be reduced by as little as one percent for this problem to be viable. It is not the aim to find the optimum solution but simply a better result than by the current method described. The Excel Solver uses a linear programming strategy.

The simplest form of this problem would be to have all the boilers available with constant efficiencies however much steam was required of them. This particular

problem could be solved using a rule-based language such as Prolog. A simple rule such as: "fill the most efficient boilers first" would work equally well, and more efficiently, than a mathematical or evolutionary technique. This was verified by results gained in the initial development of the problem. It is only when the efficiency functions become more complex that it becomes relevant to use the alternative methods, such as genetic algorithms, to be described here.

Genetic algorithms (GA) use crossover between parents and mutation to produce children. Each of these methods alone will produce convergence [5]. It was decided, in this case, to use mutation because of the complications that crossover produces. Mutation did not produce as many complications in writing the code, and those it did could be easily solved. As there was no crossover, it was deemed unnecessary to convert the numbers into binary.

The terms used in this paper are as follows:

Individual:-	Solution to the problem.
Chromosome:-	Coded solution.
Gene:-	Part of the coded solution.
Children:-	The chromosomes created as a result of crossover and mutation.
Crossover:-	Splitting two chromosomes at n points and swapping between Chromosomes to create children
Mutation:-	The altering of the gene or part of a gene to change the value of that solution slightly
Selection:-	Choosing the best children to be part of the next generation
Fitness:-	Quality of the solution as judged against the same metric
Fitness Function:-	The equations used to calculate fitness

As an additional part of the Genetic Algorithm, Memetic algorithms were used. The term "Memetic Algorithm" was first used in 1989 by Pablo Moscato [6]. The concept of the "meme" was introduced by Richard Dawkins in his book "The Selfish Gene" and is based on cultural evolution as opposed to biological evolution.

The basis for solving this problem is analogous to tipping a finite amount of fluid between containers to get the required level in each of them. However, with these containers, the liquid has a different cost value depending which container it is in and how much is in each container.

This paper is organised as follows: There is a more detailed explanation of the problem below followed by an explanation of the evolutionary techniques used in solving the problem, and how they were applied in this case. There follows a results section, conclusions drawn, and further work envisaged.

2. The Problem

The Bin-Packing Problem is the problem of filling as few containers as possible, with a selection of items, without leaving any space left over. This is considered an NP hard problem in that it cannot be solved in polynomial time. The time taken to solve the problem increases exponentially with the number of bins and items[9]. This is a class of problem that is difficult to solve using analytical or iterative techniques.

Where this problem has similarities to the Bin-Packing problem is that there is a maximum amount of steam to be generated, from a number of boilers for the least cost. The cost, in this case, being the energy needed to create the steam. It is described here as continuous in that the tonnage of steam required from each boiler is one continuous amount and not several discrete quantities.

It is required that an amount of steam is produced by several boilers attached to a header, further to the amount of steam produced, there should be some spare capacity should one of the boiler's become unavailable.

The boilers can be off, or running between minimum and maximum levels. The discontinuity, the step between the off value and minimum, makes it difficult for hill climbing techniques to find solutions to such problems. The efficiency of each boiler is calculated using a quadratic equation, which is based upon the amount of fuel needed to produce a defined amount of steam. The fuel used is calculated as the tonnage of steam divided by the efficiency for that boiler at the current capacity.

Figure 1 shows example boiler efficiency curves used in the problem formulation. The efficiencies were normalised, however, it shows that there are different shapes for each boiler. For example, some boilers are more efficient at lower capacities but not as efficient at higher values.

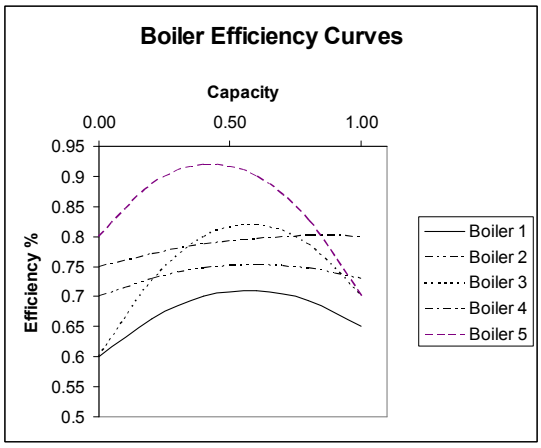

Figure 1: Efficiency Curves of Boilers

The solutions to the problem would need to include the following options:

- Any boiler can be turned on or off by the user provided the algorithm has not prohibited this.
- Allowance can be made for spare capacity.
- A check of the efficiency curves for each boiler is made.

Possible additional questions, which could be answered by the model, were, how long would it take, and, would an increase in the number of boilers significantly increase the time taken to perform the iteration.

The tools employed to solve the problem, using an evolutionary algorithm were Matlab, to perform the Genetic Algorithm, and Excel to display and collect the values. Matlab was used to generate the efficiency curves, and was used because integrated mathematical and matrix functions were needed. Excel was used because the problem was initiated in Excel and data can be displayed and saved in tables. This problem may have been posed in MDC's own software which uses linear programming and quadratic linear programming as well as other techniques for solving differential equations. Excel is used for its compatibility with the company's software.

3. Evolutionary Methods

Evolutionary Algorithms (EA) include Genetic Algorithms, ant colonisation, swarming, and memetic algorithms. These types of algorithms have been described in [4] as "Adaptive Memory Programming", because they are algorithms that include some memory features. In genetic algorithms, this occurs between iterations as the solution in the next iteration is based upon the previous search. In this paper, it is suggested that an Adaptive Memory Program should be modified in order to solve the particular problem as used in [1].

In a Genetic Algorithm, the solution population evolves from a scattered set of solutions to converge on a single solution. It will be described how this algorithm can be adapted to solve this particular problem.

In order to turn a Genetic Algorithm into a Memetic Algorithm, the GA is extended with the use of another search technique, which is performed as part of the iteration cycle. This other technique may be simulated annealing, or another hill climbing technique [7]. This other technique need not use memory.

Genetically inspired algorithms need a population, which is the starting point for the algorithm. The method used for each initial chromosome was as follows: to obtain the required amount of steam, random numbers were generated for all but one of the boilers. The maximum value for each boiler was obtained by subtracting the sum of the steam values for the other boilers from the total requirement. Should this number

be outside the allowable range, values for the other boilers are regenerated until these conditions are satisfied.

The boilers' on/off status was represented by a matrix consisting of ones and zeroes such as "11001" where one means on and zero means off. It was decided to use this method because as multiplication of anything by zero is zero. If each of the numbers in the string were multiplied by their corresponding availability, from the string "10111", this would turn the unavailable boiler off making the status string "10001". Checks were built into the algorithm to avoid situations where it was impossible to solve the problem, or where there was only one solution. For example:-

- If there was not enough or too much capacity to make the steam required plus the excess, which is needed should one of the boilers fail, there needs to be the capacity to produce as much steam as before the failure.

- Where the steam plus the spare capacity was more than the available capacity if all boilers were turned on.

- If there were only one answer to the problem, the code would break and inform the user.

Three small matrices were generated; one for whether the boiler was in use, another for whether it was available for use, and the third for if it was fixed on or off. By multiplying the matrices together, it was possible to assign the on/off status to that boiler.

Mutation was the main method of iteration. Because of the need to maintain the required capacity, swapping between individuals of the population would have been difficult.

To ensure that the initial steam values were those required, a matrix of random numbers was generated to represent the steam produced for four of the five boilers available. The amount of steam for the last boiler was the difference between the total amount of steam required and the sum of the steam produced by the other boilers. For example, if the steam required was 500 tonnes and the sum of steam from four of the boilers is 400 then the fifth boiler would be 100. If this final boiler steam requirement was a reasonable value i.e. did not exceed the maximum or was below the minimum then this string of values became an individual of the population, otherwise a further attempt was made to generate an initial value.

Where the boilers could be turned on or off, there were two methods for choosing how the steam should be re-distributed. The first method was to transfer steam from a switched off boiler to one other boiler if the capacity allowed. The second method was to redistribute the steam from the boiler with changed status randomly amongst all other available boilers.

The memetic was applied as follows: before mutation, some individuals of the population had a small amount of fuel added or subtracted from each boiler; this was then checked to see that it did not violate any restrictions, such as there not being enough spare capacity in the boiler. If the results of this change result in an improvement in the fitness function, the values are retained; otherwise they are rejected. Where the fitness function is the equation used to decide how close to the optimum the individual is.

The fitness was calculated by treating the values for each boiler as input values for the function for each boiler, the values from these functions were then summed together see equation(1). The fitness was worked out at the mutation stage using the following procedure: the population was copied and added to the second half of the matrix. The function for fitness was the minimum amount of fuel needed to generate the required amount of steam. This was calculated to be a fraction of the steam generated based upon the efficiency functions. The spare capacity was the sum of the difference between the actual amount of steam generated and the maximum that could be generated in each boiler.

$$\text{Fuel used in n boilers} = \sum \frac{x}{f(x_i)} \quad \text{where i = 1, 2, 3 …n} \qquad (1)$$

$$\text{Where} \quad \sum C_i - \sum x_i \geq S \qquad (2)$$

Where x = steam required from each boiler i, f(x) is the efficiency of the boiler to produce that capacity of steam, C is the capacity of the each boiler and S is the spare capacity of the system.

The fuel used by each boiler was evaluated from each gene, representing the tonnage of steam for each boiler; this was then summed for all individuals to find the fitness. Each individual's fitness value was then taken away from the maximum possible fitness value calculated from the all the parents and the children. The values were then sorted. The parents for the next generation were made up of the four fittest from the population, the remaining being made up of those individuals chosen by tournament selection. Tournament selection is a process for choosing the fittest individuals (ie. those as close as possible to the optimum) to be parents of the next generation. In tournament selection, n individuals are chosen at random from the population and the best goes forward. This is repeated until the population has the required number of members.

Each iteration was run until the fitnesses of half the population was equal to within six decimal places. The best individuals of each run were saved and entered as an initial population in the final run. This produced reasonably consistent results.

The algorithm is illustrated below:

3.1 The Algorithm

```
Open a link to Excel
Get data for algorithm
Generate efficiency functions

For n -1 runs
Generate first population
       Generate boiler status, on or off
       Multiply by status by availability
       Check for boiler capacity against steam required
       If capacity is more than max or less than min end as only
       one answer - end
       Generate random matrix within range of stream generated
       Multiply by status of boilers

Do until converged
Apply memetic
Mutate
       For each individual of the population if chosen
       Generate random numbers and swap between available
       boilers of same capacity
Test for Fitness
       Apply data to efficiency functions
Test for convergence
Loop
Save population

n th run:
Take fittest individuals from each run
Mutate
Test for fitness
Send result to Excel for analysis
```

4. Results

The results for the situation when all the boilers are on, and their efficiencies are assumed constant across all values, are shown below. Essentially, they would suggest selecting the most efficient boilers for the majority of the required capacity. This is a good validation check for the system.

Table 1 shows the results of running the routine for various amounts of required steam.

Table 1: Required capacity is 500 x 10^3 kg/hour, all boilers assumed to be on

Efficiency	0.65	0.70	0.75	0.80	0.85	**On**					**Fuel Used**
Iteration 1	36	36	80	148	200	1	1	1	1	1	633.77
Iteration 2	36	36	80	148	200	1	1	1	1	1	633.77
Iteration 3	36	36	80	148	200	1	1	1	1	1	633.77
Iteration 4	36	36	80	148	200	1	1	1	1	1	633.77
Iteration 5	36	36	80	148	200	1	1	1	1	1	633.77
Boiler	**#1**	**#2**	**#3**	**#4**	**#5**	**#1**	**#2**	**#3**	**#4**	**#5**	

Time to run 5 iterations of 50 cycles was 34.3 seconds
The numbers were generally converging after effectively 20 cycles

Table 2: Required capacity 450 x 10^3 kg/hour

Steam generated, boilers					**On or Off**					
#1	#2	#3	#4	#5	#1	#2	#3	#4	#5	**Fuel**
0.00	36.00	146.25	131.60	136.15	0	1	1	1	1	545
0.00	36.00	146.28	131.65	136.08	0	1	1	1	1	545
0.00	36.00	146.31	131.53	136.15	0	1	1	1	1	545
0.00	36.00	146.26	131.62	136.12	0	1	1	1	1	545
0.00	36.00	146.28	131.60	136.12	0	1	1	1	1	545

Time taken for 5 cycles was 165.48 seconds, an average of 33.1.

Table 3: Required capacity 500 x 10^3 kg/hour boiler 5 off line, using efficiency curves

Steam generated					**On or Off**					
#1	#2	#3	#4	#5	#1	#2	#3	#4	#5	**Fuel**
67.32	74.91	157.77	200.00	0.00	1	1	1	1	0	638
67.32	74.90	157.78	200.00	0.00	1	1	1	1	0	638
67.32	74.91	157.77	200.00	0.00	1	1	1	1	0	638
67.32	74.90	157.78	200.00	0.00	1	1	1	1	0	638
67.32	74.89	157.79	200.00	0.00	1	1	1	1	0	638

Time for 5 iterations, 99 seconds, an average of 19.84.

The extended example has been run with each boiler having its own efficiency curve.

4.1 Comparison

These results were then compared to results produced with an Excel spreadsheet using the Excel solver. The efficiency curves were adjusted in the GA to be the same as those in the spreadsheet solver.

The results chosen were those where spare capacity was not allowed for, and boilers could not be forced to be on or off.

Table 4: Comparison between Evolutionary Technique and Excel Solver

Steam Required	Excel Solver	Genetic Algorithm (GA)	GA + Spare Capacity
425.00	526.08	**506.34**	**515.87**
450.00	550.68	**535.96**	**544.93**
650.00	803.64	803.64	803.64
300.00	366.24	**348.41**	374.38
100.00	140.99	**114.16**	**114.16**
700.00	882.67	882.67	882.67
500.00	609.16	**598.41**	618.37
600.00	737.95	**734.53**	737.95
200.00	253.30	**232.77**	**237.84**
710.00	901.07	901.07	901.07
40.00	56.16	56.16	56.16

Those results where the GA performed better than the solver are highlighted.

The results compared were those for, initially, no spare capacity with all boilers available and none forced on. These results were the same or better than those from the spreadsheet, however, the time taken was considerably longer for the GA.

The GA failed to produce the required spare capacity where it would mean turning more boilers on. Near the limits of the capacity, the GA did not perform better than the spreadsheet solver; in one case, where spare capacity was considered, the GA produced a worse result.

The solver was able to produce results that were impractical in physical terms, e.g. when the steam generated was more than the maximum capacity of the available boilers. The code in the GA was re-programmed to not proceed where there was no practical answer. The Excel solver is a mathematical system as opposed to an algorithmic approach written for this particular problem.

5. Discussion

This problem can be solved with genetic algorithms in a relatively short space of time. Where the efficiencies are considered constant throughout the range of boiler capacities, inspection, or a rule-based system would be a quick and valid method.

The GA may not be as fast as the Solver; however, the GA is intelligent, it will not attempt to solve insoluble problems, and is more versatile in that it will allow boilers to be on or off and will perform the calculation with the spare capacity available. It has produced more efficient and more varied results than the solver.

It is hoped that the model will be extended to a more dynamic system based upon a real application. Where, for example, there is a cost for turning boilers on or off. Efficiency curves more representative of actual data could be substituted instead of the ones used. The problems could be extended to involve different fuels with different prices. This particular problem need not be restricted to power stations it could equally be applied to the filling of tanks with liquid or any problem where a static amount has to be divided into different parts, each of these parts incurring different costs.

6. Acknowledgements

The Author would like to thank Emerson Process Management, MDC Technology for posing the problem, EPSRC, MDC, and Dr Stuart McIvor and Michael Schwarz for their help in drafting this paper

References

1. Dequan Liu, Hongfei Teng, An Improved BL-algorithm of the Orthogonal Packing of Rectangles, European Journal of Operational Research, (1997).
2. Melanie Mitchell, John H. Holland: When Will a Genetic Algorithm Outperform Hill Climbing? ICGA 1993: 647.
3. Sammy Ho, Genetic Algorithms Made Easy An introduction for Control Engineers, Sunderland University, (1996).
4. É. D. Taillard, L.-M. Gambardella, M. Gendreau, J.-Y. Potvin, "Adaptive Memory Programming: A Unified View of Meta-Heuristics", Technical report IDSIA-19-98, IDSIA, Lugano, European Journal of Operational Research 135 (1), 2001, 1—16.
5. L. Kellel, B. Naudts, A. Rogers, Theoretical Aspects of Evolutionary Computing, Springer, (2001)
6. Moscato, P. "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Toward Memetic Algorithms". (1989).
7. Joshua D Knowles, David W Corne, A Comparative Assessment of the Memetic. Evolutionary and Constructive Algorithms for the Multi-objective d-MST Problem, (1998).
8. Carlos A Coello Coello, David A. Van Veldhuizen, Gary B Lamont, Evolutionary Algorithms for Solving Multi-objective Problems, (2002).
9. R.J.Hodgson, Memetic Algorithm Approach to Thin-Film Optical Coating Design, WOMA2001, (2001)
10 http://mathworld.wolfram.com/Bin-PackingProblem.html.