Genetic Algorithms for Multiobjective Scheduling of Combined Batch / Continuous Process Plants

K. J. Shaw
School of Engineering,
Murdoch University,
WA 6150,
Perth, Australia
jane@ieee.org
Tel: +44 (0) 114 2225250

P. L. Lee
School of Engineering,
Murdoch University,
WA 6150,
Perth, Australia
peter@eng.murdoch.edu.au
Tel: +61 (8) 9360 7100

H. P. Nott
CSIRO Mathematical &
Information Sciences
Private Bag 10
South Clayton MDC,
Victoria, Australia
helen.nott@cmis.csiro.au
Tel: +61 (3) 9545 8045

M. Thompson
Department of Engineering,
Sheffield Hallam University,
Sheffield, UK
m.thompson@shu.ac.uk
Tel: +(44) 114 2225250

Abstract

Systems in the process industry commonly incorporate both batch and continuous processes. These processes must be scheduled to satisfy product specifications, requirements from downstream processes and physical plant constraints. In doing so, the need to maximise various production objectives within a highly constrained environment can present an extremely difficult problem. The following paper demonstrates the difficulties of attempting to schedule the combination of discrete tasks of varying cycle time with continuous elements. Two implementations of a heuristic genetic algorithm (GA) approach are demonstrated on a processing problem that has similar characteristics to a sugar mill (Nott, 1998). The implementations include problem-specific representations, single and multiobjective approaches to handle the four problem costs, and various uses of penalty functions to avoid constraint violations. In addition, highly tailored, problem specific operators allow the GA to match its behaviour to the critical elements in the problem definition; specifically, those relating to a highly constrained shared storage facility. The results and implications of using such techniques for this type of problem are presented and discussed.

1 Introduction

The general scheduling problem of allocating a number of required tasks to particular equipment or processes can be notoriously difficult to solve. The development of genetic algorithms (GAs) as an optimisation technique for discrete event scheduling problems has been an increasingly active area of research, e.g. Davis, (1985); Bagchi et al., (1991); Yamada and Nakano, (1992); Dorndorf and Pesch, (1995); Shaw and Fleming, (1996); Mattfeld (1999). In the process industries, it is common to have systems that include both discrete event (batch) and continuous processes present. This work focuses on the development of a GA to solve such a problem.

The work is presented as follows: in section 2, a batch/continuous scheduling problem with variable cycle times is presented. This problem has been a case study for previous methods (Nott, 1998). Section 3 introduces the GA methods and describes implementations used for this problem. Section 4 describes initial experiments based upon this GA implementation, discusses the results and compares the solutions with previous methods. Finally, section 5 provides discussion and suggestions for further work on this problem.

2 Scheduling Problem

2.1 Application Description

Nott (1998) explored the problem of finding an optimal scheduling policy in sugar milling. The system includes a mixture of batch and continuous units, with interacting processes, and highly constrained operations. As in many scheduling problems, there is a high economic incentive to develop effective schedules. A simplified outline of a sugar milling system is used. The batch pan system is followed by a shared, constrained storage facility before the remainder of the downstream process, which is generally considered as operating continuously.

A profit is associated with the quantity of sugar produced, whilst processing each operation incurs a cost. Idle periods add to the cost, and drastic changes to the flowrate of the continuous units are discouraged. The presence of both batch and continuous units makes a difficult scheduling problem (Nott and Lee, 1998a). One key decision variable is the cycle time of each batch; although batch units have a fixed quantity to process, this time taken to process, known as the *cycle time*, may vary. This introduces a further element of flexibility in possible schedules compared to a problem in which the cycle times are fixed. Decisions must be made as to whether a pan should remain idle in order to benefit the overall schedule, as Goldratt, (1993), illustrates.

When both batch and continuous units are present in a process, scheduling is traditionally accomplished by discretising time and considering the problem as a job shop scheduling problem. This produces a mixed-integer linear programming problem (MILP) (Kondili et al., 1993). This work aims to apply a GA to the same problem configuration.

2.2 Problem Used in this Work

A simplified model is considered, shown in Figure 1. This consists of two batch units, which drop their output

0-7803-6375-2/00/$10.00 ©2000 IEEE.

into a single storage facility. This leads to a continuous production process, assumed to be a stream. In this system, seen in Figure 1, the cycle time of each batch may vary, within specified bounds. It is possible for the batch units to be idle. The amount in storage and the flowrate through the continuous unit are constrained. There is a profit for each unit produced in the system while a cost is incurred for each batch begun. A scheduling policy is required for a fixed period, or solution horizon, discretised into uniform time units.

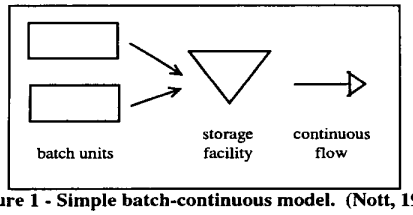

Figure 1 - Simple batch-continuous model. (Nott, 1998)

The scheduling problem is to decide the *production state* of each component of the system at each time interval, and as given by Nott & Lee (1998b).

1. Does a batch start?
2. Does a batch finish?
3. Is the unit left idle?
4. What is the flowrate through the continuous unit?

From these four decisions, the amount in storage can be deduced and the costs and profits of the scheduling evaluated. Such a problem representation is suitable for direct translation into the form of a GA.

2.3 Problem Specifications

The test problem follows specifications from Nott, (1998). Production by the two batch units must be scheduled over a period of 25 time units. The units 1 and 2 use batches of sizes 8 and 10 size units, lasting between 3-5 time units, and 4-6 time units, and costing 60 and 50 cost units respectively. Idle periods would be penalized by 100 cost units, whilst changes in flowrate would be penalized by one cost unit per flow unit per time unit. The selling price of each completed batch is 20 cost units. In addition, the storage facility may contain between 2 and 15 size units, with a starting volume of 10 size units and a variable outflow rate between 2.5 and 5 size units per time unit.

Thus the system objective is represented by:

Maximize Obj1 = PR - CB - IP - CF

where

$$PR = CC \sum_{t=ST}^{ET-1} fr_t$$

$$CB = \sum_{i=1}^{n} \sum_{t=ST}^{ET} c_i . f_{i,t}$$

$$IP = \sum_{i=1}^{n} \sum_{t=ST}^{ET} z_{i,t} CI$$

$$CF = \sum_{t=ST+1}^{ET} drf_t \qquad (1)$$

these representing the total production through the system over the entire solution horizon, the costs associated with beginning individual batches, the penalty for scheduling idle periods and the penalty for changes in flowrate respectively.

For this example:

n = number of units	i = 1,2 (n=2)
ST = start time of scheduling window	ST=1
ET = end time of scheduling window	ET = 25
pl_i = process length	$3 \le pl_1 \le 5$, $4 \le pl_2 \le 6$, pl_i integer
c_i = cost of starting batch in unit i	$c_1 = 60$, $c_2 = 50$
CI = cost of idle period	CI = 100
CC = profit of final batch	CC = 20
f_p = cost of flowrate change per unit deviation	$f_p = 1$
$s_{i,t}$ = state of batch starting in unit at time t	=1 if unit i is starting a batch at time t =0 if not
$f_{i,t}$ = state of batch finishing in unit at time t	=1 if unit i is ending a batch at time t-1 =0 if not.
$z_{i,t}$ = state of unit at time t	=1 if unit i is idle at time t =0 if not
fr_t = flowrate of continuous unit at time t	$2.5 \le fr_t \le 5$ for all t
$dfr_t = (fr_t - fr_{t-1})$ change in flowrate at time t	$0 \le drf_t \le 2.5$
OB_i = size of output batch from unit i	$OB_1 = 8$, $OB_2 = 10$ at all t
$O_{i,t}$ = output of unit i at time t	$O_{i,t} = f_{i,t} OB_i$
S_t = storage level at time t; $S_t = S_{t-1} + O_{i,t} - fr_t$	$2 \le S_t \le 15$ and $S_1 = 10$

Table 1 – Parameters used in example problem

2.4 Timing of Events

The timing of each event is critical to the satisfaction of constraints and in effective algorithm design. Each event is assumed to take effect one time unit after it is completed, batch units take effect at the end of each time unit. The timing of the batch processes and continuous flow output must be such that the storage vessel does not exceed constraints throughout the *entire continuous* time horizon as opposed to simply the end of each time interval. This specification has significant repercussions on the scheduling of the processes, as both pre- and post-release storage levels become significant. These occur just before a batch output affects the storage vessel, when the storage level is at its lowest during the time interval, and just after the batch takes effect, when the storage level is at its highest. This is illustrated by Figure 2, which shows the storage level during a time interval, *t* where both batch output and continuous flows take effect.

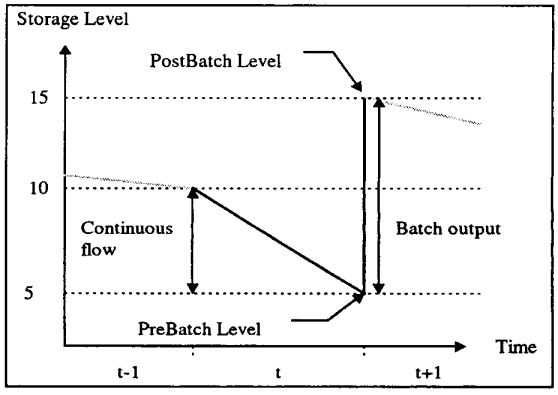

Figure 2: Effects of batch release on storage levels over time

This property is critical to successful solution of the problem. A balance must be found between a steady flowrate out of the storage unit - to meet the objective of retaining a low change in flowrate, and avoidance of constraint violation at every batch release, either by allowing the storage level to run too low or too high. The timing of the releases from the shared input of the two batch units must be arranged to avoid such constraint violation.

3 Method

Two GA methods are illustrated in this paper as attempts to solve the problem, a single objective GA (SOGA) and a multiobjective GA (MOGA). There are several advantages offered in using GAs for schedule optimisation problems. Because the method uses a population of possible solutions rather than a single point to search for potentially optimal schedules, a wide range of possibilities can be explored by searching from a range of various points simultaneously, thus finding answers which may not have been readily accessible in the past. GAs are particularly suited to the complexities of large and difficult search spaces for scheduling problems. GAs can also be of practical use for problems that cannot be tackled in exponential time. An additional advantage in a practical environment is that GAs provide 'good enough' solutions during the run such that the method can be stopped and an adequate solution to the problem found at any time in a run (Bruns, 1997). The main disadvantage of this method is that, as heuristic methods, GAs provide no guarantee of finding the optimal solution, or a bound on closeness of the final state to the optimal solution.

3.1 Multiobjective genetic algorithms

The problem definition contains a number of different costs to be optimized. It may be the case that these objectives, when regarded individually, conflict with each other or are incompatible in some way. The use of Pareto-optimality within GAs for performing multiobjective optimisation has been demonstrated in various implementations, e.g., Goldberg, (1989), Fonseca and Fleming, (1993, 1998), Horn, Nafpliotis, and Goldberg (1994), Srinivas and Deb, (1994). Its critical relevance in the solution of this problem is its ability to handle several incompatible and conflicting objectives within a single optimisation run. In addition, it is applied here to see if the alternative approaches required by the use of multiobjective optimisation offer further insights into the problem.

3.2 Development of GAs for Batch Scheduling Problems

GAs have commonly been applied to discrete-event scheduling problems, such as job shop scheduling problems. In many of these problems, the size of the batch, together with its cycle time, is predetermined, and the GA can commonly represent the schedule as a simple permutation of the available orders, with scheduling details supplemented during a 'translation stage'. For the batch scheduling problem presented, the length and timing of a batch must be decided, increasing the complexity of the problem, in the degree of decisions to be made and size of available solutions, as the problem requires sizing as well as sequencing decisions to be made. Some previous work within the field of GAs for scheduling deals with similar problems. Lee et al., (1993), explore the joint problem of lot-sizing and sequencing for a manufacturing problem. Löhl et al., (1998) present a GA designed to sequence batch operations for a polystyrene plant, which also includes continuous elements similar to those found in this problem. The inclusion of a decision on batch cycle time means that a GA must be developed which can accurately represent this information. The components of the GA are described in more detail in subsequent sections. More recently, Shaw et al., (1999) have explored the application of various MOGA implementations to benchmark batch processing problems based on the ISA S88 standards (ISA 1995).

3.3 Genetic Algorithm Representation

The GA chromosome used in this implementation consists of three distinct parts, representing the schedules for batch unit 1, batch unit 2 and the continuous flow unit. In the first two parts, giving schedules for the batch units, an integer notation is used for each element. The string length is defined by the maximum number of schedules that may be started within the time horizon considered for this application. Each batch is allocated a start and end time such that the duration of the batch lies within the prescribed limits. In order to maintain constant chromosome length, batches that are not to be used are allocated start and end times of zero. As a result of this approach batch unit 1 may be represented by two nine-element strings, representing the start and end times of the nine batches that may be scheduled over the entire time horizon. This illustrated in the example schedule shown below (Figure 3):

Batch	1	2	3	4	5	6	7	8	9
Start	0	0	0	4	7	11	14	18	23
End	0	0	0	6	10	13	17	22	25

Figure 3- Example representation

A similar notation is used to represent the schedule of Batch Unit 2. However due to the greater processing time required by Batch Unit 2, only 7 batches may be scheduled throughout the time horizon used. The continuous flow rate out of the storage vessel for each time interval is represented as a continuous variable, giving the third and final section of the individual string.

3.3.1 Objective and Constraint Definition

The genetic algorithm objective function must reflect the requirements of the system described previously. Additional care must be taken when attributing cost functions and relative weightings to each aspect of the objective as this can greatly influence performance. The representation avoids using a fixed penalty for each time interval by penalizing infeasibility by an amount relative to the amount of constraint violation. This offers the algorithm motivation over the long term to eradicate constraint violations. However, no short term

motivation is provided. As a result, an evaluation of the degree of violation is measured, and this can result in optimisation of the other objectives whilst constraint violation is effectively ignored. In order to overcome this problem a proportional term is introduced, to provide the algorithm with some means of assessing performance. An alternative to such a term is to treat the penalty as an objective in its own right to be minimised and this is discussed further in section 3.4.
At each time interval, pre-batch and post-batch storage levels are calculated and a penalty is incurred should constraint violation occur, such that:

$$total\ penalty, \qquad P_{total} = \sum_{t=1}^{t=26} p(t)$$

where

$$if\ B(t) > 15 \qquad p(t) = 20000 * [B(t) - 15]$$
$$if\ B(t) < 2 \qquad p(t) = 25000 * [2 - B(t)]$$
$$otherwise \qquad p(t) = 0$$

where B(t) is the value of storage level at time (t).

(2)

Weightings are given to the two constraint violation events in order to prevent the algorithm from simply "see-sawing" the set of chosen flow rates in an attempt to ensure that the storage level remains within the bounds such that no improvement in objective is obtained. This is a common effect observed in experimentation (Shaw et al., 1998). The objective to maximize flow rate will counter the constraint that the storage vessel must exceed a level of two units, therefore giving this constraint a higher priority, resulting in the algorithm not settling into a steady state. In addition, each individual is penalized that gives rise to constraint violation. This gives a penalty (3) for each individual. As a consequence, the overall penalty imposed upon an individual for constraint violation may be expressed as follows:

$$CV = P_{total} + 1000$$

(3)

The fixed penalty encourages the algorithm to eradicate constraint violation as opposed to simply reducing it to 'acceptable' levels. This constraint is introduced to the objectives described in (1) to give an overall objective as follows:

$$Obj1 = PR - (CB + IP + CF + CV)$$

(4)

The fifth objective is created by an infeasibility measure, discussed below (3.4).

3.3.2 Genetic Algorithm Operators

Roulette Wheel Selection (Goldberg, 1989) was used to select individuals for reproduction. Each section of the solution was considered as a possible point of crossover. The continuous flowrate section of the population used a multipoint crossover (Chipperfield et al., 1994), a standard crossover operator on the parts of two individuals representing flowrates. Mutation of the continuous flow rate used a real value mutation (Mühlenbein and Schlierkamp-Voosen, 1993). In addition, the batch elements used several operators to introduce solution change, whilst ensuring valid batch creation, such that batches of permissible duration are created.

To prevent the creation of invalid batch sequences, i.e. batches exceeding the feasible duration permissible, a number of mutation methods were created to allow new batch schedules to be created. These provide the GA with the necessary mechanism to explore the population space fully. These methods were the only operators used upon the batch schedules, providing means by which both minor and significant genetic changes may be incorporated into the individuals.

Batch Insertion

The schedule may contain a number of dummy batches. During the mutation process, if the selected batch is a dummy batch. i.e. start and end times of zero, a start time is selected at random from 1-25 and the end time is such that the minimum batch size is obtained. The schedule is then re-sequenced and each batch is evaluated to ensure that no batches overlap and that each batch is of a valid duration. This process is described in Figure 4.

```
if B > 1 ∧ ( Start_Time(B) ≤ End_Time(B - 1) ∨
(MinBatchSize < End_Time(B) - Start_Time(B) + 1 < MaxBatch Size))
{
if Start_Time(B) ≤ EndTime(B - 1)
    Start_Time(B) = End_Time(B - 1) + 1
endif

if End_Time(B) - Start_Time(B) + 1 > MaxBatch Size
    End_Time(B) = Start_Time(B) + MaxBatchSize - 1
endif

if End_Time(B) - Start_Time(B) + 1 < MinBatch Size
    End_Time(B) = Start_Time(B) + MinBatchSize - 1
endif
}
```

Figure 4: Batch Validation Procedure

This procedure ensures that only valid batch sequences are created during the mutation process. This may be illustrated by the following example:

PreMutation

Batch No	1	2	3	4	5	6	7	8	9
Start	0	0	0	4	7	11	14	18	23
End	0	0	0	6	10	13	17	22	25

Post Mutation

Batch No	1	2	3	4	5	6	7	8	9
Start	9	0	0	4	7	11	14	18	23
End	11	0	0	6	10	13	17	22	25

Validation

Batch No	1	2	3	4	5	6	7	8	9
Start	0	0	4	7	11	14	17	20	23
End	0	0	6	10	13	16	19	22	25

Figure 5 - Batch mutation process

As can be seen from the above example, this form of mutation can have a significant affect upon the individual. It has greatest significance during the early stages of the run, as this will provide significant changes to the genome and provide a powerful means by which the algorithm may explore the population space. However, as the mutation causes significant

changes it is likely to introduce damaging effects during the latter where small changes to the individuals are required. In order to introduce alterations to the batch sequencing within the genome, additional mutation methods must be introduced, these being *Start_Time* mutation and *End_Time* mutation. These methods slightly modify the start and end time of an existing batch respectively, and are discussed in the following sections.

Start_Time Mutation

Start_Time mutation first mutates the batch start time. If it gives a batch of valid duration, it is then instantly adopted. If, however, an invalid batch duration is created then the *End_Time* is altered such that a valid batch is assumed (in a similar way to that exhibited in the previous example. In each case batch validation is carried out as in Figure 4.

End_Time Mutation

This acts in exactly the same fashion as the start time mutation operator, however, in this case the *End_Time* adopts the dominant role, forcing the start time to ensure valid batch creation. The use of these mutations may allow 'shuffling' of batches away from a time point where constraint violations are present.

3.4 Multiobjective Genetic Algorithm

Multiobjective optimization abilities in the form of Pareto-based ranking are applied to the single objective GA implementation, following the method of Fonseca and Fleming (1993). Effectively the objective function is extended from the single function given in (1) to become a five objective evaluation function. The solutions are ranked by comparison of vectors of their objective values in a Pareto sense; thus one solution may excel in one objective but not another, and a second solution may offer the opposite situation. These may be ranked as equally good solutions (*non-dominated*), since neither is entirely better in all its costs than the other, i.e., neither *dominates* the other. All non-dominated solutions in the final population of the MOGA run are considered as possible contender solutions to the problem, providing multiple possibilities as to the solution until the user expresses a preference as to the relative importance of each cost's contribution to the overall goal that meets their needs best. This concept makes use of the trade-offs between incompatible solutions to preserve genuine multiobjective optimisation within the search, and contrasts with single objective optimization, in which the preferences are pre-defined, for example, by an expression such as that given in (1). It should be noted that the MOGA allows infeasible solutions to remain in the population, since the feasibility degree or penalty violation is now an objective rather than a constraint.

4 Experimental results

The GA methods were run ten times each to provide some generality of results, on a Sun Ultra SPARC 5. All GAs were encoded in MATLAB, using the GA Toolbox (Chipperfield et al., 1994). A population of 100 individuals was run for 1000 generations, using a 0.2 rate of batch validation and 0.05 mutation rate. The results were compared in terms of best values found, both for the overall objection function and for each cost individually.

4.1 Summary of resulting solutions and costs

Table 2 demonstrates the best solution found in all the runs, for the value of objective function, Obj-1, and the individual costs in (1). The best solution by the SOGA is that with maximum Obj-1 = 1413.3. The worst solution by comparison, has Obj-1 = 978.8. For comparison, MOGA values are 1320.7255 and 853.2496 for the best and worst values respectively.

Costs	Obj-1	CB	PR	IP	CF
SOGA					
Best solution	1413.3	780	2220	0	6.6513
Worst solution	978.8	730	2020	300	11.2112
MOGA					
Best solution	1320.726	1020	1999.240	300	17.091
Worst solution	853.250	920	1994.982	200	21.732

Table 2 - Values found for Obj-1 and individual costs in final population, feasible solutions only

Examining individual components of these solutions provides some insight into the search involved. The SOGA is more successful with its best solution in that it approaches the optimum value of the single objective function (1) (see section 4.2), although its worst solution does not show much improvement on previous implementation (Shaw et al, 1998). Surprisingly it still retains relatively high values for the idle period (300) in the final population; however, its attempts to limit the change in flowrate appear more successful. The overall best value found by the MOGA is not as good as the SOGA for this problem, but this might be expected given the differences in the nature of the multiobjective optimisation technique. Yet given the possible range of values found in Table 2, it still finds a competitively good solution. Rather than solely concentrating on the immediate *a priori* relationship defined in the one objective, it searches across the space of many possible combinations of the various costs. This is discussed further in 4.4; and is illustrated additionally in the much wider range of values found for all costs between the best and worst values attained. It is still capable of producing a good result and shows an increased robustness over the single objective GA.

It should be noted that infeasible solutions are retained in the population, that is, solutions which did not meet the storage constraints. Such inclusion is particularly due to the nature of the MOGA search, treating the infeasibility as an objective to be minimised rather than a penalty which eliminates infeasibility. This property is less apparent in the SOGA, as it is specifically designed to focus on the one region satisfying the particular single objective function in this way. By contrast, the MOGA finds a wide range of values to contribute to the overall single objective value. This is not necessarily detrimental to the final results of the search.

4.3.1 Schedules found

Figure 6 and Figure 7 illustrate examples of schedules found by each method. Figure 6 shows the best solution found by the single objective GA, which makes great use of batch sizes of 3. In comparison, the MOGA solution (Figure 7) preserves larger batches; as explained in 2.4 the compromise must be found between allowing a profitable number of batch releases to contribute to the production profits, and the preservation of the feasible storage levels. The inclusion of fewer, longer batches may indicate a weaker performance, as can be seen by the overall best objectives found.

Time	Unit1	Unit 2	Continuous flow rate
1			2.6503
2			2.7109
3			2.6389
4			5.0000
5			4.4039
6			4.7372
7			4.8613
8			4.9988
9			5.0000
10			4.9989
11			5.0000
12			5.0000
13			5.0000
14			4.0886
15			4.0364
16			4.8750
17			5.0000
18			5.0000
19			5.0000
20			5.0000
21			5.0000
22			5.0000
23			5.0000
24			5.0000
25			3.9641
26			

Figure 6 - Example schedule found by SOGA

Time	Unit 1	Unit 2	Flowrate
1			2.5001
2			2.5013
3			2.5126
4			4.8763
5			3.6097
6			4.9988
7			3.7363
8			3.7374
9			5.0000
10	-idle-		2.5012
11			3.8661
12			4.9887
13			3.7524
14			4.6246
15			4.7486
16			5.0000
17			3.6350
18	-idle-		5.0000
19			4.0014
20			3.8598
21			4.8738
22	-idle-		4.9864
23			4.9988
24			5.0000
25	+1	4	4.9999

Figure 7 - Example schedule found by MOGA

Figure 8 and Figure 9 show the variations in flowrates found by each method. Again, similar effects can be seen in the plotting of these variations.

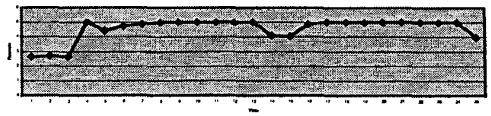

Figure 8 - Flowrate found by SOGA solution

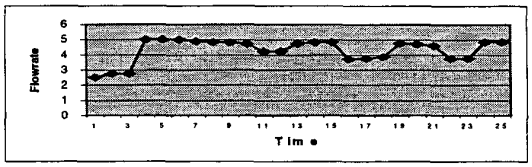

Figure 9 - Flowrate found by MOGA solution

The most successful solutions maintain a reasonably steady line to stay within the constraints of the 'feasible' storage region successfully, balancing the pre-release and post-release fluctuations discussed in 2.4. The less successful solutions fluctuate from one rate to another in an attempt to keep the storage level within feasible bounds, and suffer as a result.

4.2 Comparison with exact solution

This problem has been previously explored with a variety of possible solution methods by Nott, (1998), using mixed integer linear programming (MILP) formulations. The GA implementation considered in this paper does not find the optimum objective function value of 1415.0833, as found by the MILP. This performance was obtained using CPLEX 4.0 running on an IBM RS6000 machine. The resulting individual costs are provided in Figure 10 and the schedule itself in figure 11.

Costs	Obj-1	CB	PR	IP	CF	CCC
Values for costs	1415.08333	780	2200	0	4.917	0

Figure 10 - Optimum Obj-1 found by MILP

Time	Unit 1	Unit 2	Flowrates
1			2.6670
2			2.6670
3			2.6670
4			5.0000
5			4.8750
6			4.8750
7			4.8750
8			4.8750
9			4.8750
10			4.8750
11			4.8750
12			5.0000
13			4.3333
14			4.3333
15			4.3333
16			5.0000
17			5.0000
18			5.0000
19			5.0000
20			5.0000
21			5.0000
22			5.0000
23			5.0000
24			5.0000
25			4.0000

Figure 11 - Schedule found by MILP

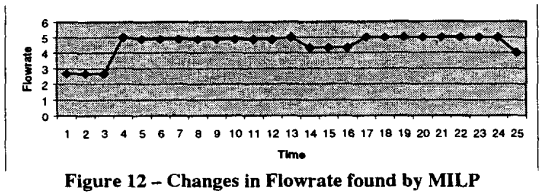

Figure 12 – Changes in Flowrate found by MILP

It is interesting to note that this solution consists of a higher number of batches with shorter cycle times (3 units), and does not schedule any batches with the longest cycle time (6 units). A key difference between this solution and that found by the GA is the value for the difference in flowrates; comparison of the individual flowrates in the schedule shows that the MILP is able to find a somewhat smoother set of values. However, this is encouraging for future GA development; adjustments may allow the real-valued strings representing this aspect to match the optimum set of flowrates once the optimum batching decisions are made. The MOGA may be used as an interactive tool, allowing the user to give priority to improving the flowrate cost at this stage in the process.

The final difference between the methods is in their runtimes; the heuristic MOGA technique searches for a mean of 115 minutes, whereas the MILP method completes its search in just under 10 (9.76) minutes, running in CPLEX on an IBM RS6000 machine. Whilst the methods have very different natures, clearly the MILP has the speed advantage in this comparison.

4.3Further Results of MOGA

As mentioned previously, the MOGA treatment allows a number of insights into the relationship of the objectives and their various effects on the final solutions. It also allows a range of solutions to be offered to the problem. By way of example, Figure 13, plots all non-dominated solutions found in one run. The x-axis enumerates the objectives, whilst the y-axis gives the values of the costs found; each line represents the vector of costs found for one non-dominated solution. The feasible solutions to the problem are those in which the fifth objective (feasibility) is zero; yet even within this category, it can be seen that there are several equally good solutions in the Pareto sense that may satisfy the problem goals. Removal of the strict linear relationship (1),(2) may allow these to be offered as alternative solutions to that given, where appropriate.

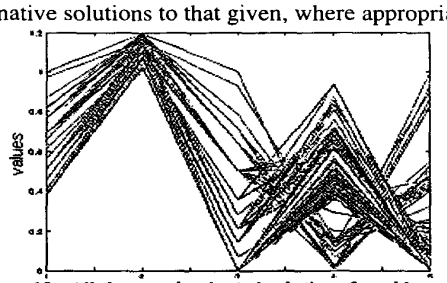

Figure 13 - All the non-dominated solutions found by one run

For example, there can be seen to be a clear trade-off in the solutions between the idle period objective (cost 3), and the change in flowrate objective (cost 4); no solution is found which has a low value for both these costs simultaneously. The relationship in (1) imposes a pressure that a zero-idle period policy should be pursued at the cost of having some variation in flowrates - and indeed proof is given that an optimal value for this relationship cannot be found with a static flowrate in Nott, (1998). By fixing the idle rate penalty at 100, and thereby deciding the relative importance of the idle penalty relative to the othe objectives, the relationship is pre-defined in this way before the search starts. However, if there *is* scope for flexibility in the choice of this penalty, then many alternative solutions are offered in the multiobjective sense which allow a lower, or even static (cost 4 = 0) solution.

A similar conflict is clear in the relationship between flowrate changes and feasibility, as has already been indicated in these solutions. Figure 13 shows those solutions that satisfy the feasibility constraint (cost 5 = 0) exactly, as well as the other solutions which are infeasible but may offer better values for all other costs. In practical terms, these solutions are only of use if the storage facility were to change its capacity! Yet in a wider context, if the user continually finds excellent solutions within this group, it may indicate a long-term suggestion as to where the improvements in the plant may be made for future reference. The MOGA can provide useful insight into such problem characteristics and suggest future approaches to the solution implementation, given the user's willingness to trade off other objective costs. Whilst the use of multiobjective optimisation within the schedule optimization means taking a different approach to the definition of the problem, it allows a more realistic approach for some real-life scheduling situation (Shaw & Fleming, 1997).

5 Conclusion

In this work, various genetic algorithm methods intended to find solutions to a mixed batch/continuous scheduling problem have been designed and implemented and their results compared to the optimal solution found by MILP. Specific implementations have considered a problem that requires decisions upon the batch durations, order of batches, the state of two discrete-process units and a continuous varying-flow unit, in a highly constrained configuration. Initial results have been presented and discussed in terms of this problem and the main requirements of the schedule optimisation process. The use of a Pareto based method indicates further insights into the problem.

Many issues are apparent for further investigation. It is apparent that the representations used are critical to the successful solution of this problem. However, it is interesting to note that this simple representation based on the start and finish times of each batch offers a concise but effective method for exploring this problem.

The trade-offs between the various costs, as seen in the varying effects of flowrate, production and idle periods, together with the feasibility of solutions, are also worth investigation. The use of linear combinations as objective functions is known to be highly sensitive to issues of weight choice; exploration of the particular contributions of the individual costs

within the objective function may be beneficial to directing the search in future implementations, as approached in section 4.3.

Many issues surround the role of constraint satisfaction within this problem. Richardson and Palmer, (1989), comment that a certain amount of infeasible solutions within the population should be encouraged, as infeasible schedules may actually contain genetic material or groups of genetic material which may, when crossed with other individuals, contribute to a very good, feasible schedule. It is necessary to decide how much feasibility should be preserved throughout the search. Effective constraint penalty functions may be designed, as shown by this GA, in order to preserve useful effects of the infeasible individuals without entirely eliminating their solutions. The MOGA avoids this issue entirely by treating infeasibility as an additional objective to be minimised. As discussed above, the results help illustrate the advantages of each algorithm.

The paper has presented GA methods of approaches to finding the optimum schedule for a challenging problem, consisting of both discrete and continuous elements. Evaluation of these methods shows that they may yet match the performance of methods that have previously found the optimum, such as the MILP implementation (4.2). In addition, this investigation has provided insight into the complexities of the problem and some of the issues that must be addressed in order to provide effective optimisation. The work has shown the significance of the problem description, constraint definition and objective costing in developing a successful solution to a deceptively difficult problem. Finally, it has shown the critical need to explore the conflicts that arise during the optimisation process between the objectives and constraints in the design of an appropriate optimization tool.

ACKNOWLEDGEMENTS

The first author gratefully acknowledges the funding of Murdoch University for this research; also the EC group at ACSE Department, University of Sheffield, for help with the GA Toolbox, and members of the School of Engineering, Murdoch University for their continued support and helpful input during this work.

References

Bagchi, S., Uckan, S., Miyabe, Y., Kawamura, K., 1991. Exploring Problem-Specific Recombination Operators for Job-Shop Scheduling, *Proc. 4th ICGA*. Morgan Kaufmann.

Chipperfield, A. J. Fleming, P. J., Pohlheim, H., 1994. A genetic algorithm toolbox for MATLAB, *Proc. Int. Conf. on Sys. Eng.*, Coventry, UK, pp. 200 - 207.

Davis, L., 1985. Job Shop Scheduling with Genetic Algorithms, *Proc. ICGA*, Lawrence Erblaum

Dorndorf and Pesch, 1995. *Evolution Based Learning in a Job Shop Scheduling Environment*, Comp. Op. Research, 22, 1, 25 - 40.

Fonseca, C. M., and Fleming, P. J., 1993. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, *Proc. 5th ICGA*.

Goldberg, D. A., 1989. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley

Goldratt, E. M., (1948). *The Goal: a process of ongoing improvement*, 2nd ed. - Aldershot : Gower, 1993.

Holland, J., 1975. *Adaptation in Natural and Artificial Systems*, University of Michigan Press

Horn, J., Nafpliotis, N, and Goldberg, D. E., 1994. A Niched Pareto GA for Multiobjective Optimisation. *Proc. First IEEE Conf. on Evolutionary Computation.*

ISA, 1995. *ANSI / ISA-S88.01.1995 Standard Batch Control; Part 1: Models and Terminology*. ISBN 1-55617-562-0, ISA.

Kondili, E., Pantelides, C. C., Sargent, R. W., H., 1993. *A general algorithm for short-term scheduling of batch operations - 1. MILP formulation*, Comp. Chem. Eng., 17 (2): 211 - 227.

Lee, I., Sikora, R., and Shaw, M. J., 1993. Joint Lot Sizing and Sequencing with Genetic Algorithms for Scheduling - Evolving the Chromosome Structure, *Proc. 5th ICGA.*

Löhl, T., Schulz, C. and Engel, S., 1998. *Sequencing of Batch Operations for a Highly Coupled Production Process: Genetic Algorithms versus Mathematical Programming*, Comp. Chem. Eng., 22, S.pp. 579 - 585, 1998.

Mattfeld, D. C., 1999. Scalable Search Spaces for Scheduling Problems, Proc. GECCO '99, pp.1616 – 1629.

Mühlenbein, H., and Schlierkamp-Voosen, D., 1993. Predictive Models for the Breeder Genetic Algorithm: 1. Continuos Parameter Optimization, *Evolutionary Computation*, 1:1, pp. 25 - 49, 1993.

Nott H.P., 1998. *Modelling Alternatives in Scheduling Mixed-Batch/Continuous Process Plants with Variable Cycle Time*, Ph.D. Thesis, School of Engineering, Murdoch University, WA. July 1998.

Nott, H.P. and Lee, P.L, (1998a). *Scheduling mixed batch/continuous process plants with variable cycle time by splitting the optimisation problem.* Fourth International Conference on Optimisation: Techniques and Applications July 1-3 Perth Australia 1998

Nott, H. P., and Lee, P. L., (1998b). *An optimal control approach for scheduling mixed batch / continuous process plants with variable cycle time*, Proceedings of the Foundations of Computer Aided Process Operations - FOCAPO, (Snowbird, Utah).

Nott, H.P. and Lee, P.L, (1999). Sets formulation to schedule mixed batch/continuous process plants with variable cycle time. *Computers and Chemical Engineering*, 23(7):875-888.

Richardson, J. T., and Palmer, M. R., 1989. *Some Guidelines for Genetic Algorithms with Penalty Functions*, Proc. ICGA3.

Shaw, K. J., and Fleming, P. J., 1996. An Initial Study of Practical Multiobjective Production Scheduling, Using Genetic Algorithms. *Proc. Int. Conf. Control '96.*

Shaw, K. J., and Fleming, P. J., 1997. Use of Rules and Preferences for Schedule Builders in Genetic Algorithms Production Scheduling. *Evolutionary Computation*: Proceedings *of the AISB '97 Workshop on Evolutionary Computation*, Springer-Verlag.

Shaw, K. J., Nott, H. P., and Lee, P. L., 1998. *A Study of the Development of a Genetic Algorithm for Scheduling Combined Batch / Continuous Process Plants*, Technical Report, School of Engineering, Murdoch University, 1998.

Shaw, K. J., Nortcliffe, A. L., Thompson, M., Love, J., and Fleming, P. J., (1999). *Interactive Batch Process Schedule Optimisation and Decision-Making using Multiobjective Genetic Algorithms*, 1999 IEEE Conference on Systems, Man and Cybernetics, Tokyo, October 1999.

Srinivas, N. and Deb, K., 1994. Multiobjective Optimisation Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation, Vol. 2, No. 3, 221 - 249.

Yamada, T., and Nakano, R., 1992. A Genetic Algorithm applicable to Large Scale Job Shop Problems, *Parallel Problem Solving from Nature*, 2, 1992, pp. 281 - 291