Using Multiple Objective Optimisation to Generate Policy Insights for System Dynamics Models

Jim Duggan,
Department of Information Technology,
National University of Ireland, Galway,
Galway,
Ireland.
Phone: 353-91-524411 Ext. 3336
Email: jim.duggan@nuigalway.ie

KEYWORDS: System Dynamics Methodology, Optimisation, Multiple Objective Optimisation, Genetic Algorithms.

ABSTRACT

Multiple objective optimisation (MOO) is an optimisation approach that has been widely used to solve optimisation problems with more than one objective function. The benefit of this approach is that it generates – using genetic algorithms - a set of non-dominated solutions which a policy maker can explore and evaluate before making a final optimal selection. This paper demonstrates that MOO can be used to assist policy makers explore a richer set of alternatives when deciding on a range of values for key parameters in their system dynamics model. In order to demonstrate the approach, a well-known case study – *The Domestic Manufacturing Company* – is used, and a stock and flow model and a multiple objective optimiser are designed and coded. The results show that valid solutions are generated, and that each of these solutions can be examined independently – and hence give greater insight into the problem at hand - before a decision is made as to the most appropriate solution.

Introduction

The aim of this paper is to demonstrate how multiple objective optimisation can be used to support policy analysis for system dynamics. The paper has the following structure: first, a summary literature review of system dynamics, optimisation, and multiple objective optimisation; second, a case study is presented based on a well-known system dynamics model, and the genetic algorithm is specified; third, a selection of experimental results are presented; and finally, conclusions and future work are described.

System Dynamics, Optimisation, and Multiple Objective Optimisation

The complimentary roles of system dynamics and optimisation are explored in detail by Coyle (1996). In this book he makes the point that "it would be highly desirable to have some automated way of performing parameter variations", and explains that, theoretically, the number of possible combinations and conceivable values of parameters is "colossal" and possibly "infinite." Within this context, some form of guided search is needed whereby good approximations to optimal solutions can be found, and Coyle employs a hill climbing heuristic algorithm to find optimal solutions.

The analogy of a hill or mountain range provides a useful way of thinking about optimisation. The most basic starting point is to consider two parameters that need to be optimised. For example, in a stock management structure problem, these parameters could be the desired stock and the stock adjustment time. When optimisation is to be performed, an additional variable must be added to the model, and this represents the payoff for a given simulation. The idea is that an optimisation algorithm will run the model many times, compare the payoffs for different parameter values, and record the combination of parameters that produces the best payoff.

Figure 1 illustrates the idea mapping parameters along with their payoff values to produce what Coyle terms the "response surface", which can be likened to a mountain range, with peaks and dips in unpredictable places. In this diagram, we have noted two

possible solutions (p2a, p1a) and (p2b, p1b). Each of these have an associated payoff, and in this case we can see that payoff at (p2b, p1b) is at a "higher altitude" than the payoff for (p2a, p1a), and so, if our goal is to maximise the payoff, (p2b, p1b) is the optimal solution.

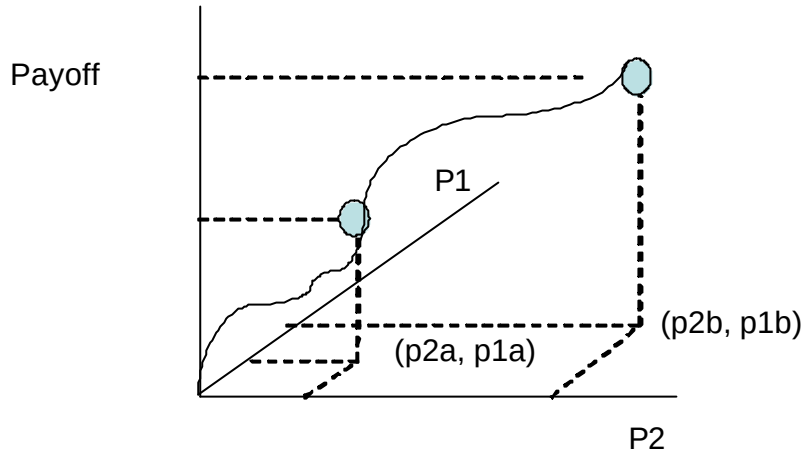

Figure 1: The hypothetical response surface for an optimisation problem with two parameters

The challenge, then, for an optimisation algorithm is to find the best possible combination of parameters that maximises (or minimises) a payoff function. To date, in addition to the work of Coyle, there have been a number of research papers that have explored the use of optimisation in system dynamics. Dangerfield and Roberts (1996) present an overview of strategy and tactics in system dynamics optimisation. Here they make a useful distinction between the two different uses of optimisation in model development. One is to use optimisation techniques in determining the best fit for historical data, while the other use (and the one employed in this paper) is "policy optimisation to improve system performance."

Keloharju and Wolstenholme (1989) present "the use of optimisation as a tool for policy analysis and design in systems dynamics models", and demonstrate their approach using the "Project Model", which was developed earlier by Richardson and Pugh (1981). According to Kelharju and Wolstenholme, one of the major benefits of using

optimisation with system dynamics models is "the saving in computational effort required by the analyst in producing policy insights in enormous." They also suggest that one of the most important issues is "that of careful selection of parameters and their allowable ranges," and that, significantly, "all values within ranges must be feasible and practical from a managerial point of view."

This concern is supported by Coyle (1996), who cautions that "it is not usually a good idea to throw all parameters into the optimiser and take the results on trust" as to do this would be to effectively "abandon thought and rely on computation." He acknowledges that the power of optimisation is "to provide a much more powerful guided search of the parameter space that could possibly be achieved by ordinary experimentation." However, he also recommends that simple experimentation is a "fundamental precursor to optimisation" and that "one cannot devise an intelligent objective function without having first 'played' with the model," and that, most importantly perhaps, "an unintelligent objective function is worse than useless."

More recently, a number of papers have been published that focus on the use of genetic algorithms for policy optimisation. Grossman (2002) comments that traditional gradient algorithms "typically fail once they have reached a local optimum" and that "genetic algorithms, if properly implemented, do not easily mistake local optima for global ones." He developed a tool that can transform High Performance System's Stella model equations into C++ code. Additional code is added that specifies an objective function, as well as marking those variables to be used as parameters.

Chen and Jeng (2004) propose a "policy design method for system dynamics models based on neural networks and genetic algorithms." Unlike other approaches, they write that their system does not need an "objective function as required by optimisation algorithms." They show how a systems dynamics model can be transformed into a partial recurrent neural network, and they effectively transform the policy design problem into a learning problem, which is solved using a combination of the genetic algorithm and neural network.

In recent years a newer form of optimisation technique has emerged: multi-objective optimization (MOO). This approach facilitates the solutions of problems with multiple objectives that "arise in a natural fashion in most disciplines and their solution has been a challenge to researchers for a long time" (Coello Coello 2002). Deb (2001) writes that "in the parlance of management, such search and optimisation problems are known as multiple criteria decision making (MCDM)", and he explains that multiple objective optimisation problems can also be reduced to a single weighted objective function.

However, the disadvantage reducing these problems to a single weighted form is that "unless a reliable and accurate preference vector is available, the optimal solution obtained by such methods is highly subjective to the particular user." Therefore, the key benefit of MOO algorithms is that the process of finding the optimal solution is less subjective, because "a user does not need any relative preference vector information." The idea is that what Deb (2001) terms "higher-level information" can be applied to the problem once the set of optimal solutions are calculated, where this higher level information is based on the skill, judgement and experience of the decision maker.

In MOO, the idea of *dominance* plays a key role in arriving at these optimal solutions. To illustrate this, consider figure 2, which graphs a set of possible solutions to an optimisation problem. In this case, the decision maker wants to purchase a house, and the two objectives are (1) to minimise costs and (2) to minimise the distance from the city centre.

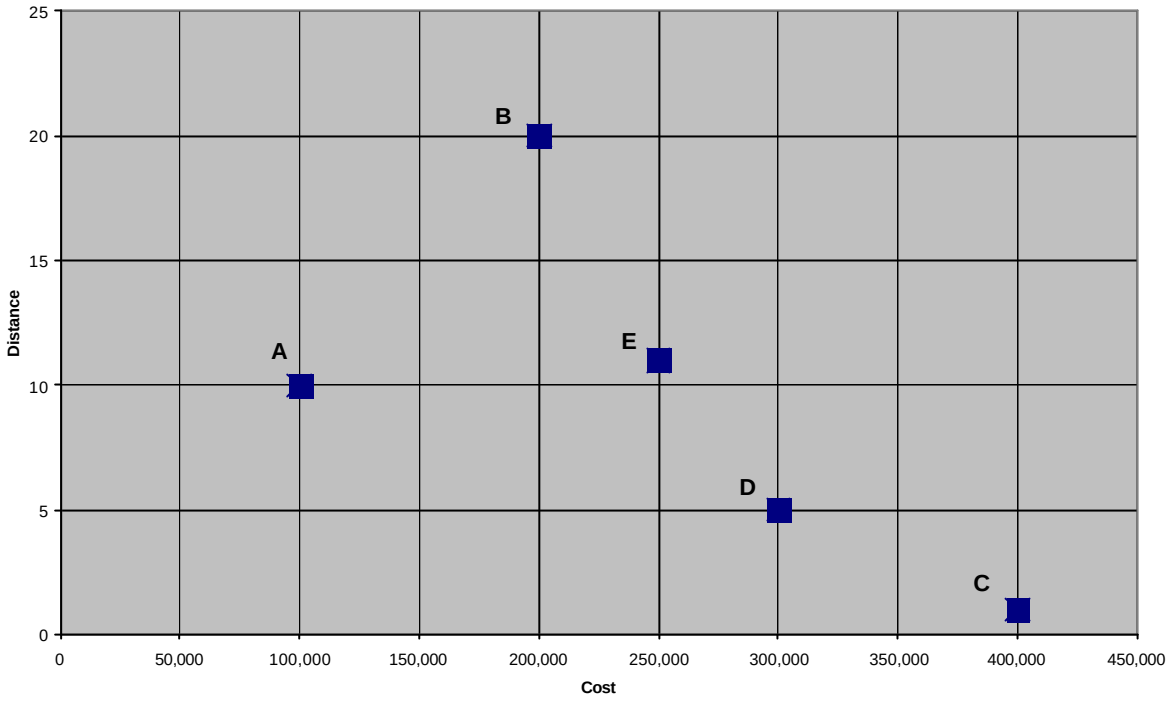

Figure 2: Set of conflicting alternatives for a house purchase scenario

By comparing all solutions, we can construct a set of solutions that are not dominated by any other solution – this is called the Pareto-optimal set. A solution is not dominated when it cannot be bettered by another solution on both objectives. For example, in our house-purchase decision:

- Solution A is non-dominated because no other solution exists that has a lower cost and a shorter distance to the city centre.
- Solution B is dominated because there is a solution that has both a lower cost and is closer to the city (i.e. Solution A).
- Therefore, A is a better solution to B, and it will get a higher ranking in the solution process.
- By going through a similar analysis of the other solutions, the non-dominated set can be composed, and will be made up of three solutions: A, D and C. A more detailed working of this solution process is contained in the case study section.

The MOO approach can now be applied to any system dynamics model where more than one objective needs to be optimised, and where a selection of parameters and their suitable ranges can be specified. There are many examples of such scenarios. For example, in the classic beer game model (Sterman 1989), each actor is trying to minimise their costs, and if a supply chain coordinator has the task of trading off these objectives, then it could be formulated as a multiple objective optimisation problem. Similarly, in a project management scenario for a software company, a policy maker might be interested in exploring the different range of solutions that would involve minimising the often conflicting objectives of time to market and defect ratio.

System Dynamics and MOO – A Case Study

The example is divided into two parts: First, the stock and flow model is described, including the payoff functions that are used in the optimisation process. Second; the multiple objective optimisation algorithm that performs the optimisations is presented.

The Stock and Flow Model

In Coyle's (1996) textbook on System Dynamics, he presents a variety of case studies, including one for the Domestic Manufacturing Company (DMC), which manufactures washing machines, and this example forms the basis of our experiment with multiple objective optimisation. The features of this case study are:

- Customers tend to order large batches of machines, with a delivery required six weeks after an order.
- Each machine requires a supply of raw materials, and fresh supplies of raw materials take six weeks to arrive.
- Demand is unpredictable, and the DMC tries to maintain the backlog down to a target level.
- The production rate is set using two factors. First, the discrepancy between actual and target backlog is eliminate over a four week period. Second, to keep up with

the current order level, the production rate includes the expected order rate for the coming week.

- The backlog target is six times the smoothed average orders (over four weeks).
- The target raw material level is based on smoothing production variations over 4 weeks and aiming to have sufficient stocks to cover 8 weeks of average production.
- Raw materials stock is kept to its average level by ordering enough to cover discrepancies between the actual and target, and also ensuring that there is sufficient raw materials to copy with expected raw material demand.

A Stock and flow representation of this model is shown in figure 3. It comprises two stock management control structures, one to manage the backlog and production starts, the other to manage the acquisition of raw materials.

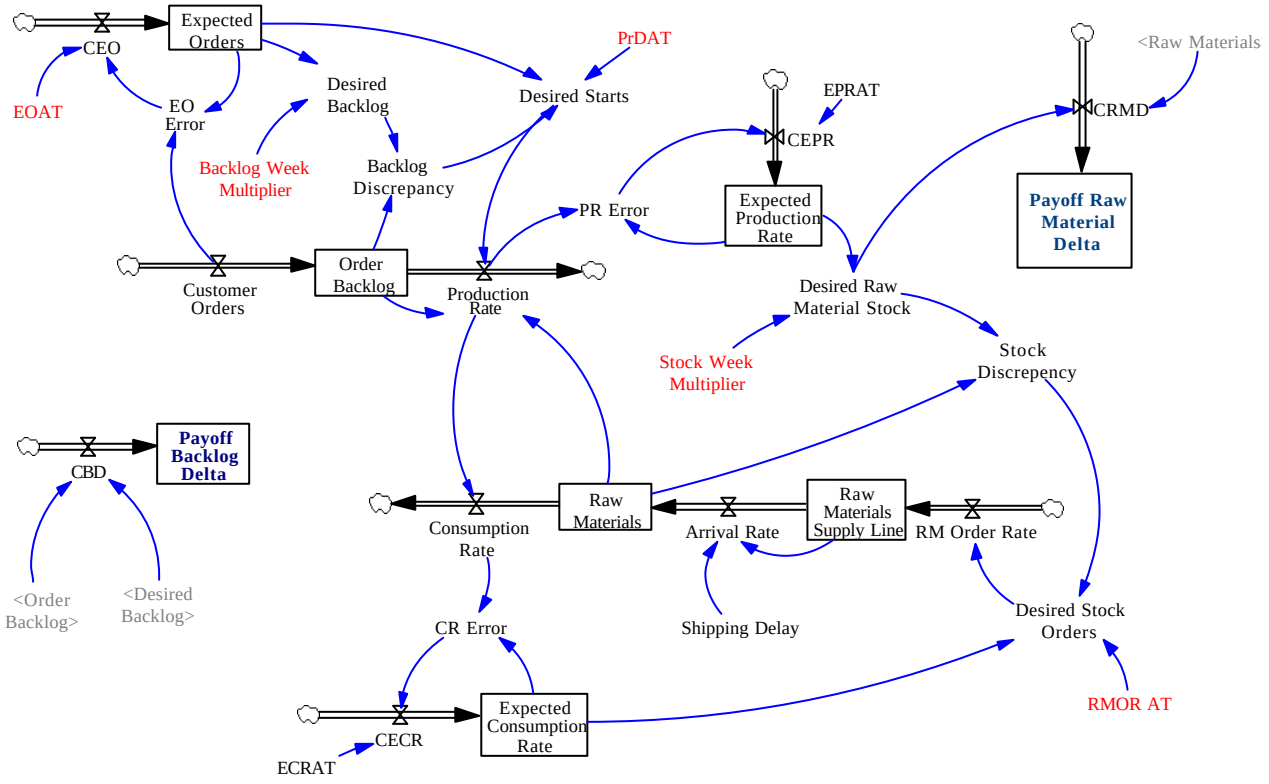

Figure 3: The stock and flow model for the Domestic Manufacturing Company

For the backlog management aspect of the model, the equations are:

(1)	Order Backlog = INTEG(Customer Orders - Production Rate , 120)
(2)	Customer Orders = Exogenous variable, can vary from run to run
(3)	Production Rate = min (Desired Starts , min (Order Backlog , Raw Materials))

The Order Backlog (1) is the integral of the Customer Orders minus the Production Rate. Customer Orders (2) are shown as constant, but as with any exogenous variable, this can vary for any given run of the model. The Production Rate (3) is determined by a stock management structure, albeit constrained so that there will be no starts if there are not enough raw materials present or the Order Backlog is less than the Desired Starts.

Desired Starts (4) is the sum of Expected Orders (8) and the Backlog Discrepancy (5) adjusted by the production adjustment time (12). Expected Orders are based on a smoothing of Customer Orders, where the adjustment time is defined in equation (11).

(4)	Desired Starts = max (0, (Backlog Discrepancy / PrDAT) + Expected Orders)
(5)	Backlog Discrepancy = Order Backlog - Desired Backlog
(6)	Desired Backlog = Backlog Week Multiplier * Expected Orders
(7)	Backlog Week Multiplier = 6 *(optimisation parameter)*
(8)	Expected Orders = INTEG(CEO , 20)
(9)	CEO = EO Error / EOAT
(10)	EO Error = Customer Orders - Expected Orders
(11)	EOAT = 4 *(optimisation parameter)*
(12)	PrDAT = 1 *(optimisation parameter)*

Desired Backlog (6) is based on the value specified in the problem narrative, and is a product of Expected Orders and Backlog Week Multiplier (7).

The raw materials management section of the model is made up of the following equations.

(13)	Raw Materials = INTEG(Arrival Rate - Consumption Rate , 0)
(14)	Consumption Rate = Production Rate
(15)	Arrival Rate = Raw Materials Supply Line / Shipping Delay
(16)	Shipping Delay = 6

Raw Materials (13) are an integral of the Arrival Rate (15) minus the Consumption Rate (14). The Consumption Rate equals the Production Rate (3), and assumes a one-to-one correspondence between orders and raw material consumption. The Arrival Rate is a first order delay on the Raw Materials Supply Line (17), divided by the Shipping Delay (16). The reason for choosing a first order delay was to simplify the actual coding of the model at a later stage, although a higher-order or pipeline delay may well be more appropriate in real-life circumstances.

(17)	Raw Materials Supply Line = INTEG(RM Order Rate - Arrival Rate , 2)
(18)	RM Order Rate = Desired Stock Orders
(19)	Desired Stock Orders = max (0, (Stock Discrepancy / RMOR AT) + Expected Consumption Rate)
(20)	Stock Discrepancy = Desired Raw Material Stock - Raw Materials
(21)	Expected Consumption Rate = INTEG(CECR , 20)
(22)	RMOR AT = 4 *(Optimisation Parameter)*
(23)	CECR = CR Error / ECRAT
(24)	CR Error = Consumption Rate - Expected Consumption Rate
(25)	ECRAT = 4

The Raw Materials Supply Line (17) is an integral of the Raw Material Order Rate (18) minus the Arrival Rate (15). The Raw Material Order Rate is simply the Desired Stock Orders (19), which is based on a stock management structure based on the Expected Consumption Rate (21) and the Stock Discrepancy (20), adjusted by the RMOR AT (22).

Working further back through the decision logic for the Desired Stock Orders of raw materials, a number of equations remain. First, the Desired Raw Materials Stock (26) is a product of the Expected Production Rate (27) and the Stock Week Multiplier (28).

(26)	Desired Raw Material Stock = Expected Production Rate * Stock Week Multiplier
(27)	Expected Production Rate = INTEG(CEPR , 20)
(28)	Stock Week Multiplier = 8 *(Optimisation Parameter)*
(29)	CEPR = PR Error / EPRAT
(30)	EPRAT = 4
(31)	PR Error = Production Rate - Expected Production Rate

Finally, as one of the model's goals is to be used to optimise the set of parameters (7), (11), (12), (22) and (28), and allow the user to trade-off between two different objectives, and a set of payoff functions are needed so that these can be minimised. [Note: there is no theoretical limit on the number of objectives that can be optimised. Two are selected for readability purposes so that they can be compared and studies by the reader in two dimensions.]

The first payoff function is the Payoff Backlog Delta (32), which keeps track of the cumulative squared distance of the Order Backlog (1) from the Desired Backlog (6). The assumption here is that in this control system, the closer the actual is to the goal (over time), the better the result.

(32)	Payoff Backlog Delta = INTEG(CBD , 0)
(33)	CBD = (Desired Backlog - Order Backlog) * (Desired Backlog - Order Backlog)

The second payoff function is the Payoff Raw Material Delta (34), which similarly keeps track of the squared distance of the Desired Raw Materials Stock (26) and the Raw Materials (13).

(34)	Payoff Raw Material Delta = INTEG(CRMD , 0)
(35)	CRMD = (Desired Raw Material Stock - Raw Materials) * (Desired Raw Material Stock - Raw Materials)

To summarise, for a given parameter set, equations (32) and (34) are effectively the output of the simulation, and these are used to calculate the payoff for each objective in the multiple objective optimiser. Therefore, the two objectives for the optimisation can be stated as:

Find the optimum values of the parameters:

- Backlog Week Multiplier
- Expected Order Adjustment Time (EOAT)
- Production Starts Adjustment Time (PrDAT)
- Stock Week Multiplier
- Raw Materials Order Adjustment Time (RMOR AT)

That minimises the values:

- Payoff Backlog Delta
- Payoff Raw Material Delta

This is the goal of the multiple objective genetic algorithm, which is now detailed.

The Multiple Objective Optimisation Algorithm

MOO algorithms usually employ genetic algorithms (GAs), which have been widely used to solve optimisation problems in many different domains. GAs apply the Darwinian-related concepts of selection, crossover and mutation on an initial population of random solutions. Over many generations, these solutions converge towards the "best fit", where the strong solutions survive and the weak ones are "weeded out". In effect, this process is used to discover the peaks (or optima) for each combination of parameter. The mutation operator enables new random points of the solution space to be explored, and is designed to ensure that solutions do not converge too quickly to local optimum points.

The first step in developing a GA is to decide on the representation of the solution. In many examples, bit strings (ones or zeros) are used, but integers and real numbers may also be employed. Solutions are typically represented by a one-dimensional array. Returning to our case study, the representation of a solution for this case is a one dimensional array of five places, where each location represents a value for one of the parameters – in each case a real number in the range [1.0-15.0]

P1	**P2**	**P3**	**P4**	**P5**
Backlog Week Multiplier *Equation (7)*	EOAT *Equation (11)*	PrDAT *Equation (12)*	RMOR AT *Equation (22)*	Backlog Week Multiplier *Equation (28)*

The task of the optimisation process is to discover the best combination of value parameters that optimises the two payoffs described in equations (32) and (34). To illustrate how the process works, let's assume that five solutions are randomly generated, and each solution contains a value for each parameter. These initial solutions are shown below.

Solution	P1	P2	P3	P4	P5
1	1.4	8.2	14.7	4.9	7.2
2	5.3	1.1	9.7	14.1	4.2
3	4.2	8.9	1.4	1.7	2.1
4	7.3	11.8	10.8	1.9	3.2
5	4.9	1.9	2.7	1.7	14.9

The next step in the GA solution process is to evaluate the payoff for each of these solutions, and this can only be accomplished by running the simulation model specified earlier (equations 1 – 35) for each of the parameter sets contained in a given solution. This also means that the process is computationally intensive, as the simulation model has to be run for each possible solution (normally 100) for every generation (also normally 100), so that 10,000 individual simulations are needed to arrive at the optimum.

When the simulation is complete, only two results are of interest to the optimiser, and these are the values of the two payoff functions, each representing an objective to be optimised (equations 32 and 34). In this sample illustration, we assume that, for each individual simulation of a solution, the payoff values are returned, as shown below in the next table.

Solution	P1	P2	P3	P4	P5	Payoff Backlog Delta	Payoff RM Delta
1	1.4	8.2	14.7	4.9	7.2	37.12	150.98
2	5.3	1.1	9.7	14.1	4.2	220.0	65.0
3	4.2	8.9	1.4	1.7	2.1	150.12	85.18
4	7.3	11.8	10.8	1.9	3.2	78.65	110.8
5	4.9	1.9	2.7	1.7	14.9	100.0	20.8

This payoff values are now critically important for the GA, as they are a measure of the fitness of each solution. In this case, both our objectives are minimisation, so our next task is to compare our solutions to see which ones are non-dominated (i.e. not bettered on both counts by any other solution). As explained earlier, non-dominated solutions are given the highest ranking, and this ranking is then used to calculate their probability of selection for the next generation. It is useful to represent each solution graphically on a scatter diagram, where each payoff is represented on a separate axis.

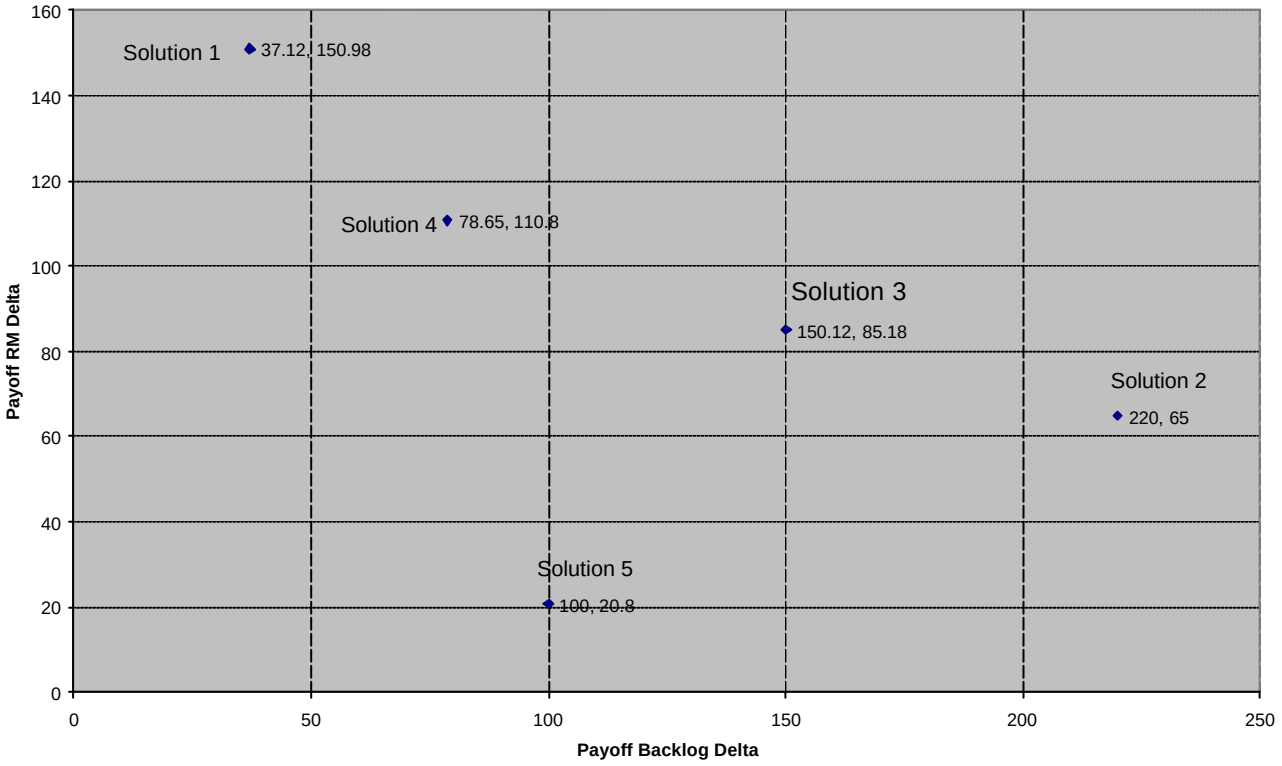

Figure 4: Determining whether solutions are not dominated

The dominance relationships are shown in the next table, and should be read from left to right, i.e. solution 1 does not dominate solution 1, solution 1 dominates solution 2, etc.

	Solution 1	**Solution 2**	**Solution 3**	**Solution 4**	**Solution 5**
Solution 1	Does not dominate	Dominates	Dominates	Does not dominate	Does not dominate
Solution 2	Is dominated by	Does not dominate	Does not dominate	Is dominated by	Is dominated by
Solution 3	Is dominated by	Does not dominate	Does not dominate	Is dominated by	Is dominated by
Solution 4	Does not dominate	Dominates	Dominates	Does not dominate	Does not dominate
Solution 5	Does not dominate	Dominates	Dominates	Does not dominate	Does not dominate
Overall	Dominated	Dominated	Not Dominated	Not dominated	Not dominated
Ranking	2	2	5	5	5

For these five solutions, three are non-dominated, so they share the highest ranking, while the two dominated solutions are ranked joint lowest (because, when taken as a pair of solutions, solutions 1 and 2 do not dominate each other.) As a general rule, the highest ranking is determined by the actual population size, which is normally 100.

This purpose of the ranking process is to assign probabilities for selection to each solution. This is because the next stage of the GA process is called selection, and this involves randomly selecting solutions from the current population, where the probability of selection is proportional to their ranking (i.e. the higher the rank, the higher the probability of being selected). The selection probabilities for a given solution is calculated as follows:

Selection Probability (Solution i) = Rank (i) / Sum (Rank(1)… Rank(N))

Where N is the total number of solutions (in this example N=5).

Solution	Payoff Backlog Delta	Payoff RM Delta	Is Dominated	Ranking	Probability of Selection for New Generation
1	37.12	150.98	NO	5	5/19
2	220.0	65.0	YES	2	2/19
3	150.12	85.18	YES	2	2/19
4	78.65	110.8	NO	5	5/19
5	100.0	20.8	NO	5	5/19

Based on this ranking process, the strongest solutions (i.e. those with the highest ranking) have the highest probability of "survival" when the next generation of solutions gets selected. Within the algorithm, a technique known as roulette wheel selection is used, where a random uniform number between [0,1] is selected, and, depending on where this falls on the "wheel", the appropriate solution is selected and placed in a new set known as the *mating pool.* For example, if the random number was in the range [0, 5/19], then solution 1 would be selected and placed in the mating pool.

Once the selection process has occurred, two important transformations then take place which result in a modification of the mating pool. The first of these is called *crossover*, where a proportion of solutions are selected and "crossed-over" in a pair-wise manner, and the "offspring" then replace their parents in the solution set. A random crossover point is selected. Each solution is "spliced" at this crossover point, and the first part of one solution is combined with the second half of the other, and vice versa.

To illustrate this point, if solutions 1 and 5 had been selected for the mating pool, and if they then had been marked for crossover, these solutions would then be considered to be "parents."

Solution	P1	P2	P3	P4	P5
1	1.4	8.2	14.7	4.9	7.2
5	4.9	1.9	2.7	1.7	14.9

Furthermore, if we assume that the crossover location was randomly selected as location 3, the crossover process would give rise to two new solutions, which would then replace their parents. We label these 1' and 5'.

Solution	P1	P2	P3	P4	P5
1'	1.4	8.2	14.7	1.7	14.9
5'	4.9	1.9	2.7	4.9	7.2

These are new solutions, and during the next generation of the GA, their fitness will be decided on by running the simulation model, and ranking the solutions based on whether or not they are dominated by others.

Finally, before the next generation can be processed, the final transformation, called *mutation*, is applied to a small proportion of solutions (normally less than 0.05). For a given solution selected for mutation, one of its parameters will be assigned a new random value from the range [1-15]. For example, if solution 1' was selected for mutation, first one of its locations would be randomly selected, and then that location would be randomly changed to a new value. For example, if the location selected was 3, and the new value was 1.2, then solution 1' would have the following values:

Solution	P1	P2	P3	P4	P5
1’	1.4	8.2	1.2	1.7	14.9

When the mutation stage is completed, the mating pool now becomes the new population, and the process of ranking, selection, crossover and mutation is repeated for up to 100 generations, and the solution will converge to produce a set of solutions that comprise the Pareto front. These are then the optimal solutions, and the decision maker can analyse these before selecting the most suitable alternative (based on the decision maker’s judgement and priorities). A sample set of results is documented in the next section, but before that, the overall algorithm for the multiple objective GA is summarised.

```
RandomisePopulation();

for(genNumber = 0; genNumber < numberGenerations; genNumber++)
{
      CalculatePayoffs();

      RankAllSolutions();

      CalculateSelectionProbabilities();

      GenerateMatingPool();

      CrossoverMatingPool();

      MutateMatingPool();

      CopyMatingPoolToPopulation();
}

SaveParetoFront();
```

Experimental Results

A sample solution set is now explored. In figure 5, the entire solution set is shown first, and the cluster of points near the x and y axes is an indication of how the solutions converge over time. While the initial generations would contain solutions that are dominated, through successive generations of selection, crossover, and mutation, a final Pareto set emerges that contains the set of optimal solutions.

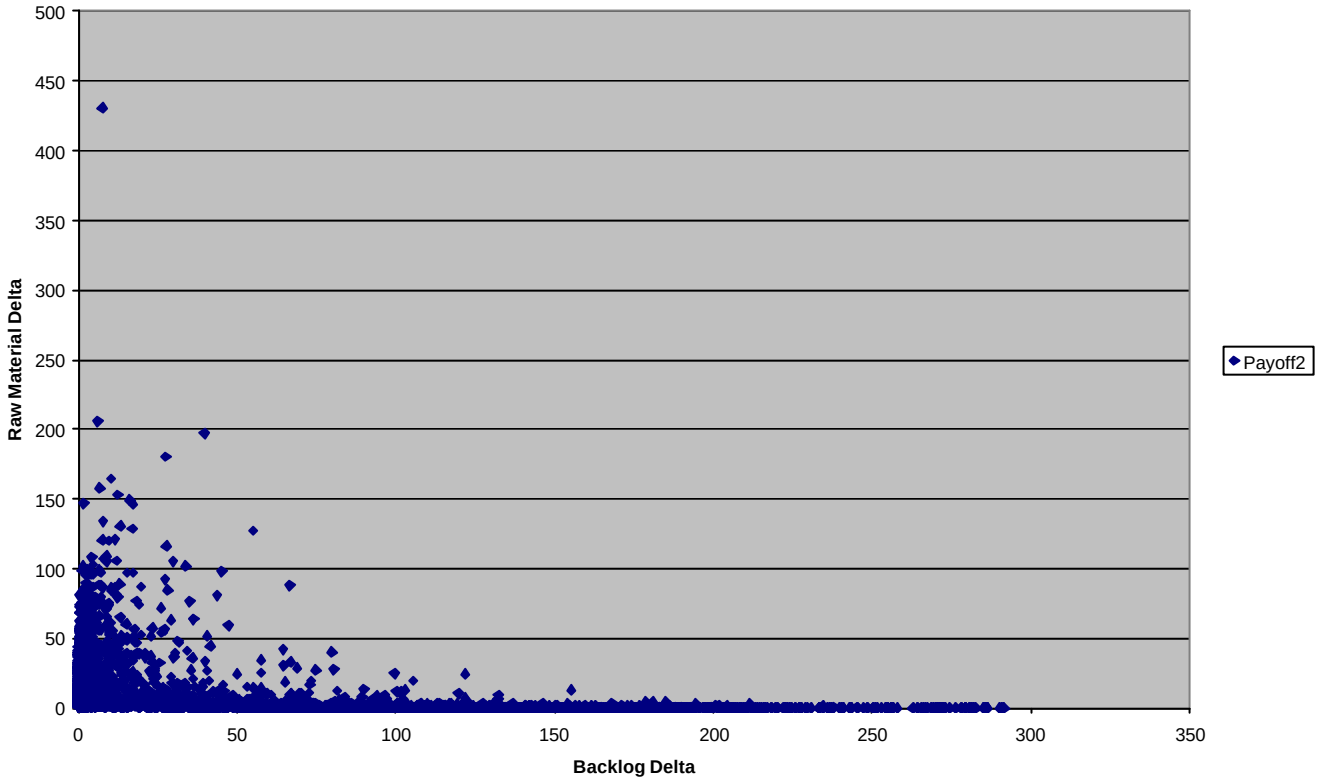

Figure 5: The complete set of solutions produced by an optimisation run where parameters remain constant

After the final generation, the set of non-dominated solutions is generated, as shown in figure 6. Each solution consists of the optimal values of the parameters that were used to run the simulation model. In theory, any one of these solutions can be selected by the decision maker.

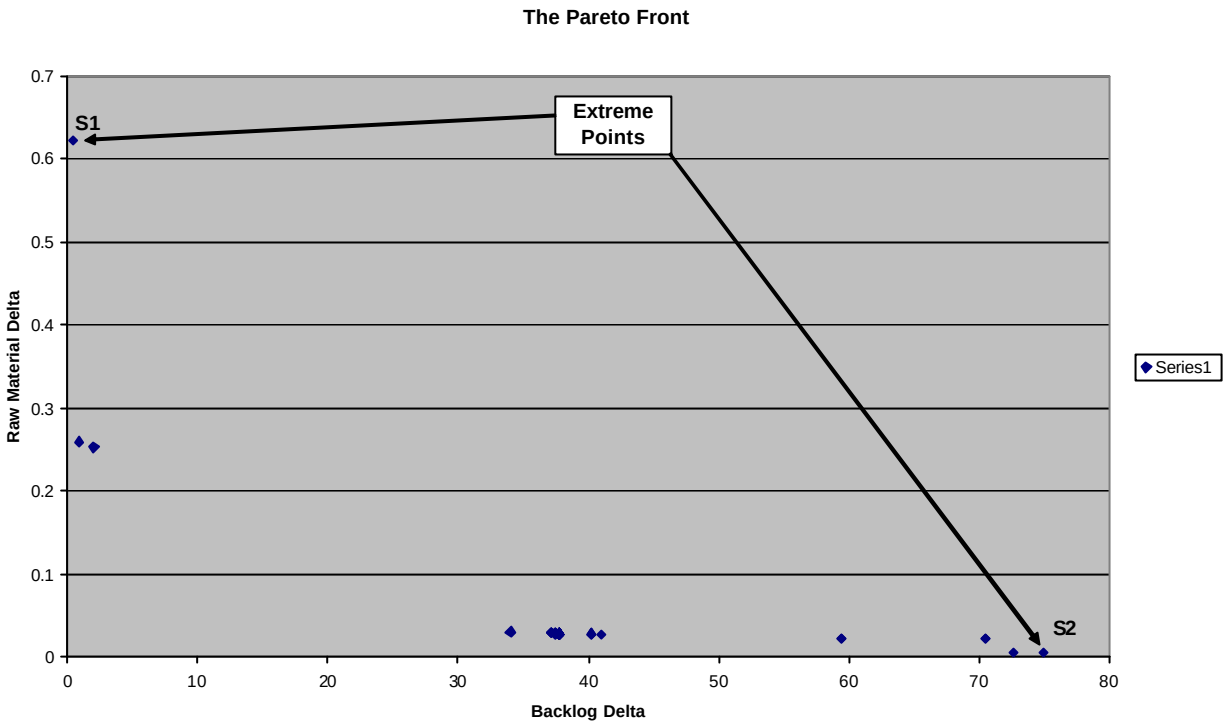

Figure 6: The Pareto front for an optimisation run

We now consider two solutions from the optimal set, where these solutions are the two extreme points. Solution 1 (S1) has coordinates (0.522, 0.622), while solution 2 (S2) has the values (74.94, 0.005). [These numbers are scaled down using a payoff reference value similar to the approach used by Coyle's (1996) optimisations.] These selected solutions have the following characteristics:

- S1 performs best on the Backlog Delta payoff function. This means that if the decision maker selects this solution, they effectively give greater importance to having the backlog closer to its goal than having the raw material value closer to its corresponding goal.

- S2 performs best on the Raw Material Delta payoff function. This means that if the decision maker selects this solution, the decision maker places the minimisation of raw material variability as a higher priority than order variability.

Further information may then be obtained on each solution, and these are shown below.

Solution	**P1** *Eqn (7)*	**P2** *Eqn (11)*	**P3** *Eqn (12)*	**P4** *Eqn (22)*	**P5** *Eqn (28)*
S1	14.21	4.73	11.03	1.11	1.39
S2	14.21	6.46	11.03	1.11	1.39

Interestingly, in this case, the only parameter value that makes the extreme solutions different is P2 (Expected Order Adjustment Time – EOAT), but it would be difficult to prove any causal relationship here. Additional solutions from the Pareto front from this particular optimisation run are summarised below.

BacklogWeekMult P1	EO_AT P2	PrD_AT P3	RMOR_AT P4	StockWeekMult P5	Payoff1	Payoff2
13.96	9.69	7.62	1.11	1.40	59.44577	0.021629
14.21	6.47	11.03	1.11	1.02	37.78219	0.027973
14.21	6.47	7.62	5.27	1.40	37.78219	0.027973
14.21	6.47	11.03	1.11	1.40	37.19301	0.02889
13.96	9.69	11.03	3.24	1.40	37.78219	0.027973
8.08	6.47	11.03	1.11	1.02	37.78219	0.027973
13.96	9.69	11.03	1.11	2.42	37.19301	0.02889
14.21	6.47	10.21	1.11	1.40	37.78219	0.027973
13.96	9.69	11.03	1.11	1.40	0.905484	0.259402

An assumption made in the models to date was that the value of a parameter does not change over the course of the simulation. The underlying GA structure was designed in such as way so that the values of parameters can change over time. This is achieved by increasing the size of the solution so that it can hold parameter data for each time interval. For example, if we were to run this five parameter an optimisation for 50 time units, our solution array has 5 x 50 = 250 elements. The shape of the overall solution set, shown in figure 7, is markedly different to the pattern from the earlier run where the optimum parameter values remained constant for the entire simulation.

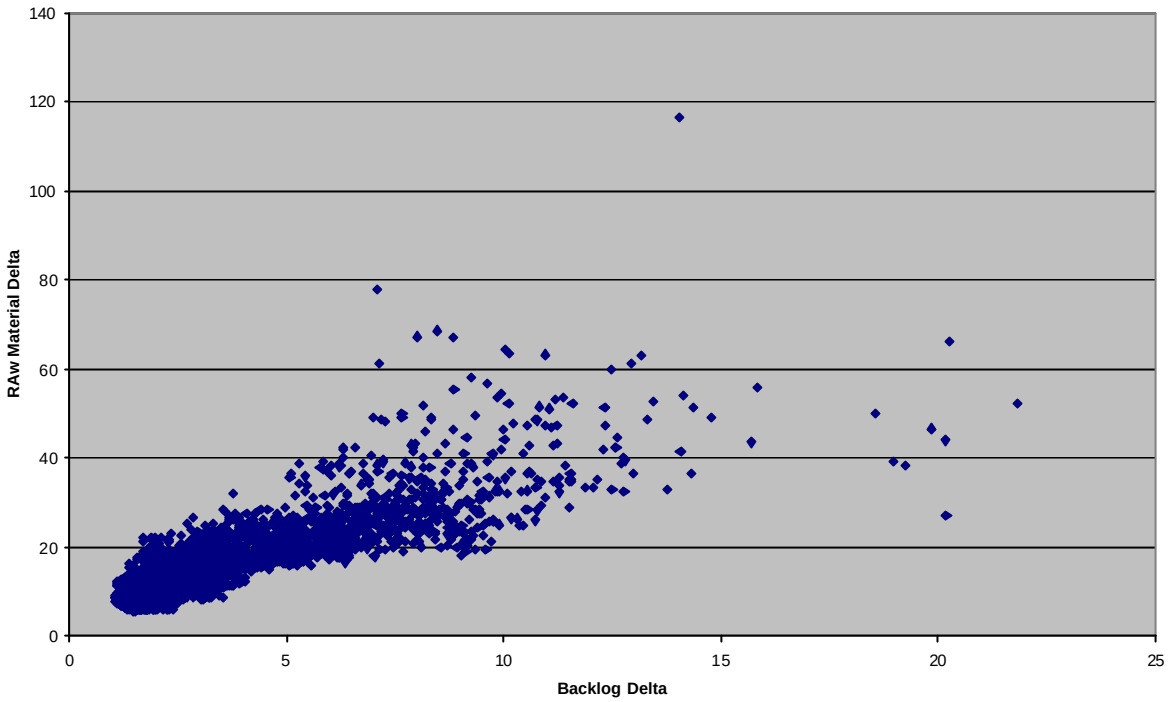

Figure 7: The Pareto front for an optimisation run where parameters change every time unit

In comparing figure 7 with figure 5, it is clear that:

- The overall solutions in figure 5 produce results that are better on both objectives.
- The solutions in the earlier optimisation run have a greater spread and are more diverse, and so will give the decision maker a great degree of choice is trading off the different solutions.

Further work needs to be done on the possible reasons for this significant difference, but on the face of it, it does suggest that having constancy in the choice of parameters – for this model – does lead to better payoffs. This observation seems to confirm Coyle's (1996) view that "it is a rule of thumb in control engineering that reducing gains and increasing delays is likely to increase stability." Sample output from an optimisation run where parameters change frequently is shown in figure 8.

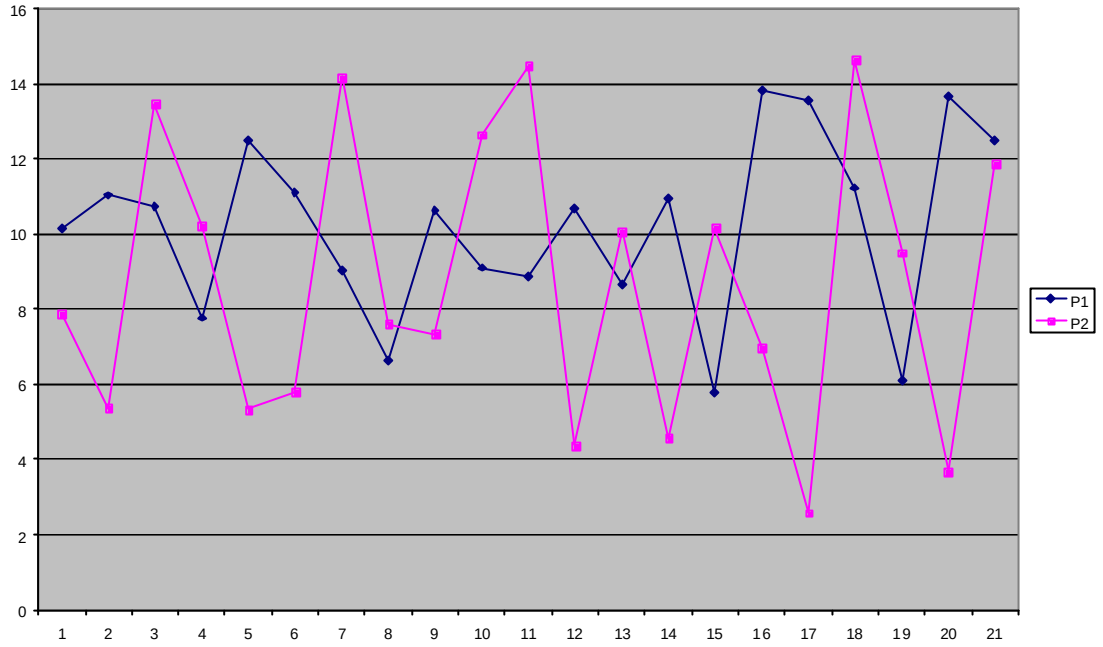

Figure 8: An optimisation run with frequently changing parameter values

Conclusions

The aim of this paper was to demonstrate that multiple objective optimisation can be employed with system dynamics models in order to assist decision makers select their preferred optimal solution. A sample model, based on the Domestic Manufacturing Company was implemented, and tests showed that valid and feasible results were generated. Future work will extend the application of this technique to a new set of problems, for example, those involving the classic trade off of time, defects and cost, and it is hoped that these future models will also provide a useful basis to evaluate whether or not that having constancy in the values of parameters leads to better overall payoffs. Another research challenge is to incorporate more efficient and effective genetic algorithms, through the use of elitism, archiving and more advanced forms of crossover.

References

Dangerfield, B. and C. Roberts. 1996. "An Overview of Strategy and Tactics in System Dynamics Optimisation." *Journal of the Operational Research Society,* 47, pp 405-423.

Chen, Yao-Tsung, and Bingchiang Jeng. (2004). "Policy Design to Fitting Desired Behaviour Pattern for System Dynamics Models." *Proceedings of the 22nd International Conference of the System Dynamics Society,* Oxford, England.

Coello Coello, C.A., Van Veldhuizen, D.A., and Lamont, G.B. 2002. Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York, NY 10013.

Coyle, R.G. 1996. *System Dynamics: A Practical Approach.* Chapman and Hall, London, UK.

Deb, K. 2001. *Multi-Objective Optimisation Using Evolutionary Algorithms.* John Wiley and Sons, Baffins Lane, Chichester, UK.

Grossman, B. 2002. "Policy Optimization in Dynamic Models with Genetic Algorithms" *Proceedings of the 20th International Conference of the System Dynamics Society,* Palermo, Italy.

Keloharju, R. and E.F. Wolstenholme. 1989. "A Case Study in System Dynamics Optimisation." *J. Ops. Res. Soc.,* Vol. 40, No.3., pp 221-230.

Richardson, G.P and A.L Pugh III. 1981. *Introduction to Systems Dynamics Modeling with DYNAMO.* MIT Press.

Sterman, J. 1989. "Modeling managerial behaviour: Misperceptions of feedback in a dynamic decision making experiment." *Management Science.* 35 (3), pp 321-339.