Optimisation techniques for a computer simulation of a pastoral dairy farm

Rohan P. S. Hart[1], Michael T. Larcombe[2], Robert A. Sherlock[3] and Lloyd A. Smith[1]

[1] Department of Computer Science, University of Waikato, Hamilton, New Zealand.
las@cs.waikato.ac.nz

[2] Maffra Herd Improvement Co-op, 2 Foster St, Maffra, Victoria 3860, Australia.

[3] Dairying Research Corporation Limited, Hamilton, New Zealand. sherlockr@drc.co.nz

Address for correspondence:

Dr. R.A. Sherlock
Dairying Research Corporation
Private Bag 3123
Hamilton 2001
NEW ZEALAND

Abstract

This paper compares different methods of optimising the management variables in UDDER, a commercially-available computer simulation model of a pastoral dairy farm. The emphasis is on identifying the best optimisation strategy for this complex multi-dimensional system, taking the simulation model as a given constant. The optimisation methods studied are based on significantly different principles, with differing strengths and weaknesses: two hill-climbing algorithms (Nelder-Mead Simplex and Powell's Direction Set), and a genetic algorithm. Rather than examine all facets of dairy farm management, a single problem is optimised — that of maximising milkfat production while maintaining the health of the herd and pasture.

The results show that while the genetic algorithm can determine good regions within the search space quickly, it is considerably slower than either hill-climber at finding the optimal point within that region. The hillclimbers, in contrast, are fast but have a tendency to get trapped on local maxima and thus fail to find the true optimum. This led to the development of a hybrid algorithm which utilises the initial global search of the genetic algorithm, followed by the more efficient local search of a hill-climber. This hybrid algorithm discovered near-optimal points much more quickly than the genetic algorithm, and with more reliability than the hill-climber.

Keywords: Optimisation; Dairy farm; Computer model; Genetic algorithm

1 Introduction

The optimisation of many real world problems is too complex, time consuming and/or expensive to be carried out on the physical system itself. Computer simulation provides an inevitably simplified view of the processes involved, but allows algorithmic optimisation methods to be applied at relatively low cost.

Computer simulation and optimisation has the potential to improve dairy (and other) farming practices without the need to physically test enormous numbers of different strategies. One problem for farmers is that it is difficult to determine whether a farm with no obvious problems is as good as it can get, or whether there exists a different management scheme which would provide more profit. Even if a better

scheme does exist, if the current farm is at or near a local optimum then any gradual change towards a better farm will likely result in an initial decrease in profitability. More significantly, the extremely large number of permutations of farm mangement parameters, the inevitable high level of 'noise' from climatic variation between seasons and the difficulty of performing well-controlled field trials (even in research institutes), means that optimisation by exhaustive search is totally impractical. Given a good farm simulation model (not a trivial requirement) however, algorithmic optimisation has a very low cost and, even if far from perfect, should considerably narrow the range of parameters and provide a tight focus for field experimentation.

However, models of sytems such as dairy farms have high dimensionality and complex response surfaces. The optimisation problem in such cases is non-trivial. In this work we investigate the performance of some optimisation schemes with a specific commercially available dairy farm computer simulation, UDDER 8.0 (Larcombe 1989). In fact UDDER itself provides an optimisation system based on the Simplex algorithm (Nelder & Mead 1965), but the robustness of this relatively simple algorithm with such a complex problem is open to question.

We take the pragmatic approach of evaluating different optimisation strategies for the UDDER dairy farm simulator 'as given'; ie no changes are made to the model itself. Three additional optimisation methods are adapted to integrate with the UDDER model. The first is a more powerful local gradient based algorithm, Powell's Direction Set method. The second is an example of a global search method, a genetic algorithm. The direction set method theoretically converges faster to the optimum than does the Simplex method, which would allow larger areas of the problem space to be searched in the same time frame. The genetic algorithm, on the other hand, always searches the entire problem space before narrowing down to the good regions. This ability should allow the global optimum to be found, no matter how different it is from the current farm. The results of these experiments suggested a third algorithm which starts with a genetic search to determine the good regions of the problem space, and then quickly narrowed in on the optimum within that region using the direction set method. Experiments comparing this new 'hybrid' algorithm to the others are also described. The paper is based on the thesis of Hart (1996).

2 The UDDER Dairy Farm Simulation

The UDDER simulator models a single dairy. It predicts the flow of energy within the dairy production system by calculating the consumption and output of energy by different classes of cattle on the farm.The collection of relations that form the energy-balance model of the processes involved in the entire farm are updated over each ten day period through a complete year. In most cases these relations are based on piecewise linear and exponential functions least-squares fitted to experimental data.

The energy model is based on processes determined experimentally at the Macalister Research Farm (Maffra Herd Improvement Co-operative, Maffra, Victoria, Australia) and Ellinbank Dairy Research Institute (Department of Food and Agriculture, Victoria, Australia), and on recommendations of the Agricultural Research Council (1980) concerning the various energy requirements of a cow's metabolism. The model is given to the user as a modifiable structure with certain predetermined limits, and includes the best current estimates of relationships, based on experimental data.

There are four main stages in the simulation. The first stage determines the number of cows of each type[1] (milkers at various stages of lactation, calves, yearlings and dry cows), the second determines the available pasture, the third calculates amount of energy consumed from pasture and other supplementary feeds, and the last determines resulting milk production and changes to the cows' condition (amount of body fat).

The number of cows of a particular type is determined from the initial number of animals on the farm, the calving pattern, drying-off strategy, cow sales and the proportion grazed on the home farm.

Farm grazing land is subdivided into 50 equal virtual paddocks, each of a specific grass height (or, equivalently, pasture cover in kgDM/ha). Paddocks can be used for grazing, fodder conservation, or fodder crops. Growth in each virtual paddock over each ten day period is based on a predicted rate of growth obtained experimentally for a given herbage mass per hectare. This growth curve is specified once

[1] Refer to the Appendix A for definitions of these types.

per month, then scaled for each ten day period to give a smoother transition between months. As the grass growth is dependent upon density, a more accurate estimate can be calculated by dividing the ten day periods into smaller portions and determining the growth during each sub-period, taking into account the changes in herbage mass from the previous sub-period.

The amount of pasture offered to cows is calculated based on grazing management (area of pasture offered per cow per day, specified by rotation length), herbage mass, and pasture characteristics such as digestibility and density. Digestibility is predicted from a decreasing function of the herbage adjusted for the time of the year.

The potential intake of pasture is determined from the cow's stage of lactation, day of the year, live weight, body condition score, and the calculated digestibility. This is then reduced to take into account the effects of availability¾ at times the cow's intake will be restricted by the height of the pasture. The quantity of pasture eaten if there are no supplements provided is then determined. If supplements (concentrates or fodder) are provided the quantity of pasture that is replaced is then determined relative to the actual mass eaten—the more mass eaten, the more replaced by one of the substitutes.

The last variable affecting the energy that the cow takes in from pasture is the digestibility of the pasture eaten. When a cow has a reasonable selection of pasture she will eat the more digestible parts first, increasing the average digestibility of the pasture intake. This increase depends on the amount of selection possible, and is a non-trivial function of pasture digestibility and the proportion of actual pasture intake to that originally predicted. This digestibility is used in the calculation for energy gained from pasture eaten; more digestible pasture will give a greater part of its energy to the cow.

The calculated energy intake is then divided among the different processes within a cow—for maintenance, pregnancy, growth, milk production and body condition. The energy used in maintenance, pregnancy and growth is based on a specified standard, while milk production and changes to the body condition score are determined by a complex relationship based on the net energy intake, the cow's condition score, the concentration of energy in the diet, and the stage of lactation. This is represented by an empirical relation derived from experimental data. After the condition scores of all the cows have been updated, the above calculations are repeated for the next ten day period.

3 Optimisation Schemes

In this section a brief overview of the optimisation problem is given, followed by a description of the optimisation algorithms used in this study and details of their implementation. The interface to the core of UDDER was made through source routines for Menu Definition, Fitness Function and Optimisation Functions.

3.1 Overview

Optimisers are normally described as working in a *search space* defined by the set of all possible inputs to a problem-specific function. A specific set of input variables describes a point within this multi-dimensional space, and the function returns the result value at that point. The values of all points in the search space form a results surface and it is on that surface that the algorithm looks for the optimum. In a more complex system the problem function may produce multiple outputs, and thus there are multiple results surfaces. Treatment of such cases is very problem-dependent, and in this work discussion is restricted to single-output systems.

Algorithms applicable to general black-box systems[1] can be divided broadly into two categories: local and global search. Local search algorithms take a single point in the search space and look for a better one nearby. Once a better point is found, the algorithm searches for an even better point near the new one. This cycle repeats until there is no longer a better point in the local area. Local search algorithms are frequently referred to as *hill-climbers* because their operation is determined by the local gradient. (In the special case of systems where functional derivative information is available, these algorithms are considerably simplified. However, that is not the case with the present problem). Global search algorithms, in contrast, try to obtain an overall view of the search space before narrowing down to the

[1] Systems which accept an input and produce an output, without giving access to internal workings.

optimal point.
Some knowledge of the specifics of a problem is useful when deciding how it is best optimised. For example, if the problem is smooth and continuous over the entire input domain, with few local optima, then a simple hill-climber is likely to be a good choice. However, when the search space is discontinuous, contains many local optima, or has regions for which there is no output value for some inputs, then a more robust global optimisation scheme is likely to be more appropriate. In such cases appropriate use of heuristics may simplify the optimisation surface. eg by hiding invalid regions of the search space.

3.2 Evolutionary and Genetic Algorithms

Evolutionary Algorithms (EA's) comprise a general class of algorithm used to solve optimisation and machine learning problems. They are based on ideas first put forward in the biological field by Darwin (1899) who formulated the hypothesis that natural species evolve in reaction to their environment so as to optimise their fitness. This complex and subtle process (eg see Dawkins 1996) is transformed into a useful algorithm by removing all but the most basic of biological evolutionary properties. These are:

i) a diverse population of individuals, each having a particular set of genes (a *genome*) which determines the individual's fitness in the given environment;

ii) a modification operator that generates new individuals having a mixture of genes from two (or possibly more) 'parents', thus providing a way to explore new areas in the search space;

iii) a selection operator which restricts the reproduction of less fit individuals.

Optimisation by EA's takes place by allowing successive population generations to evolve by 'reproduction' constrained by the above rules ii) and iii). After a sufficient length of time a fairly stationary population containing some optimal individuals should result.

The Genetic Algorithm (GA) is a form of Evolutionary Algorithm with the following additional properties (Goldberg, 1989):

i) the problem to be solved can be described by an underlying *fitness landscape* characterised by some collection of variables;

ii) an individual's genome is a string of characters from which the problem variables can be decoded;

iii) there are two types of modification operator: a crossover operator which mixes the genetic material from a pair of individuals when creating a new individual[1], and a mutation operator which acts on a single individual's genome to create occasional random changes.

The fitness landscape is a specific type of results surface. It is characterised by having a single *fitness* value for each point in the search space. The mapping of the input variables to the fitness landscape is described by two functions:

simulation(input variables) ⇒ simulation results

fitness function(simulation results) ⇒ fitness

In real world problems the simulation result is usually a vector whose components may have very different scales and units of measure. They may also conflict with one another. The fitness function must balance all those things to produce a simple scalar numerical result. Finding a 'correct' definition of the fitness function necessary to optimise the desired problem is generally non-trivial. For the problem examined in this work however, the fitness was simply taken as the number of kilograms of milkfat produced by the farm each year. An example of a two-variable fitness landscapes from this problem is given in Fig. 1.

[1] A simplified version of what occurs during sexual reproduction.

Crossover and mutation perform two distinct tasks in the search for better solutions. The crossover operator provides exploitative search in that it has the potential to group the best genes from the parents together to form a new individual that is better than either parent. Mutation, on the other hand, provides some level of random local search so that the population does not become trapped into converging to a suboptimal solution. It allows individuals to escape local optima, as well as allowing the possibility of recreation of good genes that have been lost.

An important property of the GA is that it does not rely on any special properties of the underlying fitness landscape because no gradient information, implicit or otherwise, is required or used. That independence allows its use in a wide range of difficult problems such as those which are NP-complete[1] (Garey & Johnson, 1979).

The most basic genetic algorithm is the Simple Genetic Algorithm (SGA) defined by Goldberg (1989). This has a fixed population size of fixed length individuals whose genome is a concatenation of binary strings (genes) which are the system variables in encoded form. The births and deaths of individuals are organised into synchronous generations. In each generation the adults are replaced by their children; this process occurs at the same time for all adult individuals. Single-point cross-over is used (Fig. 2), while mutation randomly flips bits in an individual's genome. The crossover point is chosen randomly for each new crossover. An individual's fitness is calculated based on its genome, and involves no other problem specific knowledge. In other words, the raw fitness is not changed to reflect any knowledge about the interactions between the input variables, or deflect the population away from known local minima.

The structure of the Simple Genetic Algorithm is shown in Fig. 3. Each iteration through the **while** loop is one generation. Because the optimum is not usually known, the stopping condition is commonly met when there is no improvement over the best individual within the last *N* generations, where N is equal to one hundred for the problems examined. Another alternative is to stop when a majority of the population has converged to some small group of genomes. For the problems examined this was 95% (or 90% if the population was smaller than twenty).

3.3 Genetic Algorithm Implementation

There are three main design choices which affect any GA: variable encoding (the genetic representation of the optimisation variables), the definition of the fitness function, and constraint violation (eg how to handle genomes that produce no result). The overall design of the GA optimiser for this work followed the traditional SGA model described by Goldberg (1989), with modifications to this structure to include more advanced crossover and selection schemes. These aspects are now discussed in some detail.

3.3.1 Variable Encoding

For the UDDER simulator, some of the input variables are real valued. For all of the variables used in the experiments, a discrete version was constructed if necessary. The resolution of the discrete versions was determined by examination of the simulation code, and in all but one case the variables had hard-coded resolutions. Thus a discrete binary encoding was used. Had more of the variables been truly real valued then real-valued encoding might have been more appropriate.

3.3.2 Fitness Function

The UDDER simulator allows modelling of dairy farm economics based on such factors as feed, stocking rate and milk prices. Because a farm's main economic output is milk production, it was decided that this alone would provide an adequate measure of the farm's profitability for the purposes of this work. ie the farm's fitness was defined simply as the weight of milk-fat produced over the entire milking season. The variables which are highly dependent on assumed prices, such as grain supplements and fertiliser, were removed from the optimisation parameter set, remaining fixed at the initial values for the default farm. Thus the goal was one of providing the greatest milkfat output within the default farm's original economic framework by optimising the internal management variables.

---

[1] Problems for which no polynomial time solution is known, i.e., a linear increase in problem size gives an exponential (or greater) increase in the time to required to solve it.

3.3.3 Constraint Violation

For some combinations of input variables the UDDER simulator indicates that all the cows have died due to underfeeding. The simplistic solution would be to assign zero fitness to such farms. However, there was evidence that these regions were close to the optima, and hence had some 'good genes' which would be lost in such a procedure, thereby aversely affecting the convergence of the GA. The alternative strategy adopted was to give such farms a very low (but non-zero) fitness, which ensured that any good genes were not removed automatically from the population pool while still providing a large incentive away from such solutions. (Had the simulator given information as to how quickly the cows died, thus indicating a measure of 'badness', this could have been advantageously incorporated in a graded fitness measure).

We note that allowing even the worst individuals a small chance to live may unfortunately increase the danger of 'lethals'. Such individuals contain the exact opposite of good building blocks (which increase fitness) by containing genes which cause most other genomes to lose fitness, or to exceed the constraints, thereby 'killing' good genes. Because the probability of selection is based on fitness, any crossover involving a lethal is most likely to be with one of the best current individuals. One strategy to overcome the problem of such 'lethals' is to compare the fitnesses of the two children from the crossover with that of their parents, and keep the best two individuals from this set of four. One negative point for this scheme is that it discriminates against movement across a valley in the optimisation surface, possibly restricting the search power of the algorithm. Another is that this strategy is in essence a more extreme version of 'keep the best alive', which shows premature convergence to sub-optimal points. For these reasons no modification to the GA was explicitly included to combat lethals.

3.3.4 Selection Schemes

The most basic version of the SGA uses roulette selection. This simply adds all of the fitnesses together, and assigns each individual a probability of being chosen for the current crossover based on its percentage of the total sum of fitness. Thus the average individual in a population of 100 will have a 1% chance of being chosen, while the best will have a higher value. Because there are 100 selections made, the average individual is likely to be involved as one parent in one crossover in that generation, while the best may be in two, three or more.

However, this scheme breaks down as the population begins to converge, with the variation of fitnesses decreasing to some small value. When that happens, each individual has an almost equal chance of being selected, and the search becomes essentially random. To address this problem, the fitnesses are scaled before determining the total fitness sum, and hence their individual chances of being chosen. This is normally a simple linear scaling so that the maximum fitness becomes twice the average. This means that the best fit individual will be involved in two crossovers in a generation, on average. In some cases this scaling can not be achieved without shifting some fitnesses below zero, so an alternate scheme must be used that involves finding a scaling which maps the minimum fitness to zero while keeping the average unchanged. In such cases the maximum will be greater than twice the average, and the best fit individual is likely to be involved in more than two crossovers in a generation. Either way, the effects on early generations are negligible, or reduce the range of fitness, while providing the expansion necessary as the population converges, to allow further directed search to occur.

The reduction of fitness range in early generations can be useful in that it reduces the chance that any extremely fit individual will dominate early on. Such domination would mean that the individual is involved in an unbalancing number of crossovers, which usually results in premature convergence. In other words, the very fit individual and its descendants take over the entire population very quickly, essentially restricting the search to a small subset of the entire problem space.

There are, however, theoretical difficulties involved in the use of scaled or unscaled roulette selection. Because each individual's chance of selection is determined by some numerical scaling of its fitness value, such properties as *reproduction rate*[1] and *selection intensity*[2] are impossible to determine for the scheme as a whole. That is because the properties are dependent on the specific fitness distribution in

[1] The ratio of the number of individuals with a certain fitness before and after selection. Good individuals should have a higher reproduction rate than poor ones.

[2] A measure of the change in the population's average fitness. Higher selection intensities cause faster convergence.

each generation. Blickle and Thiele (1995) give a full discussion of the properties of roulette selection (where it is called proportional selection) and a comparison between it and other schemes.

Due to the above difficulties another scheme, tournament selection, was investigated. Because this is based on the ranking of the fitnesses in the population and not on the actual fitness values, the properties of population change can be modified without the need for specialised scaling. A tournament of size N selects N *different* individuals from the population, each individual having an equal chance of being included. From those N individuals, the fittest is used as a parent in a crossover operation, the others being ignored. Changing the selection intensity requires only increasing or decreasing the number N. As N increases, the weighting shifts towards the fitter individuals, while a decrease in N causes more random selection.

One side-effect of tournament selection (and ranked selection schemes in general) is that the N-1 worst individuals will never be selected for crossover. Thus the very bad individuals, which were given a small probability of survival in roulette selection on the off chance they contained good genes, are excluded from breeding. In this work the value N = 2 was used for the smaller populations (20 or 10 individuals), thus excluding only 5 or 10% of the population. The larger populations (fifty individuals) used N = 3, and hence excluded only 4%.

3.3.5 Crossover operators

The operators used in this work were the two-point and uniform crossovers. Two-point crossover is an extension to the one-point crossover which removes any bias by treating the genome as a circular, rather than linear, structure. Two points are then chosen on the circle, and the segments swapped between the parents (Fig. 4). This eliminates the bias of the SGA's one-point crossover (described in Section 3.2 and Fig. 2) arising from the fact that in this scheme the left- and right-most bits of an individual's genome are guaranteed to come from different parents.

Uniform crossover selects each bit in a child with equal probability from either parent. The advantage of this scheme over two-point crossover is that children can be more different from their parents. Ignoring the effects of mutation, with two-point crossover there will always be a sequence of bits (or a single bit) which are exact copies of the same sequence in a parent. This means that two-point crossover allows the search to reach only certain sections of the search space (dependent on the parents' locations), while uniform crossover has less restriction as to the child's search area.

The major apparent disadvantage with uniform crossover is that it doesn't follow the guidelines set down in the 'building-block hypothesis' (Goldgerg, 1989). Because there is little chance of sequences of bits from either parent being copied whole into a child, any building blocks that may exist will be broken up, unless both parents contain identical copies of that section. With two-point crossover the chance of losing a correct building block is only 2/L, where *L* is the length of the genome. This is offset somewhat by the fact that the order of genes in the genome is not important when using uniform crossover. Ordering is important for two-point crossover, because for the building block hypothesis to work well genes which are closely related should be placed close together so that they are more likely to remain together.

From experimental evidence (De Jong & Spears 1990), using uniform crossover for "small" population sizes (in comparison to the problem complexity) may be better due to the increased disruption. The idea behind that is that more disruption in the small population stops it from converging too quickly, and allows a wider range of individuals to be examined. Conversely, "large" populations work better with two-point crossover, assuming that the ordering of the genes is good.

3.4 Hill-climbing Algorithms

All hill-climbing algorithms implement some form of sequential local search. ie from a starting point P1 they search the immediate vicinity for a point P2 which is more nearly optimal. If found, an even better point P3 in the vicinity of P2 is sought, and so on until no further improvement is achieved. The optimisation algorithm provided with UDDER is a hillclimber¾the well-known Nelder-Mead Simplex method (Nelder and Mead, 1965; Press *et. al.*, 1992). It is taken as the basis for comparison with the other algorithms developed in this work.

3.4.1 Direction Set Methods in Multi-dimensions (Powell's Algorithm)

Powell's algorithm is essentially a way to use one-dimensional minimisation (or maximisation) algorithms on multi-dimensional problems. The algorithm (Press *et. al*. 1992) is a specific implementation based on the *direction set* idea, which is to take N (the dimensionality of the search space) linearly independent, mutually *conjugate vectors*, and apply a linear maximisation along each. In other words, the algorithm performs a maximisation for N one-dimensional *slice* through the search space.

Conjugate vectors are used because they have the property that a maximisation along one is not penalised by the following maximisation along another. For the two vectors, **u** and **v**, to be conjugate requires that, after a maximisation along **u**, the gradient of the function being solved stays perpendicular to **u** as we maximise along **v**. As the gradient will be perpendicular to **u** at the maximum along the slice defined by **u** (because the gradient is zero in the **u** direction at that point), this requires that there is no change in the gradient perpendicular to **u** when moving along **v**. Calculation of conjugate vectors therefore requires determination of the function's second derivative. Also, if the function is quadratic then each direction needs to be maximised only once to reach the global maximum. Thus the algorithm *converges quadratically* if a set of linearly independent and mutually conjugate directions can be found (Press *et. al*. 1992).

For the more usual case (including the present problem) where derivatives for the result surface cannot be analytically determined, Powell's method tries to build a conjugate set of directions during the optimisation. It starts with a set of linearly independent directions, usually the dimensional axes. From there, the algorithm attempts to determine a general direction of maximisation. For example, Fig. 5 shows how Powell's algorithm might find the maximum of a simple two-dimensional surface (represented as a contour plot). When started at point P1 (at the centre of the hexagon) it first looks for the best point along a line parallel to the horizontal axis. The range covered by this first search is shown by the double-headed arrow. The point P2 is the best point in that range, so the algorithm searches in the vertical axis direction, resulting in the point P3. Now that the algorithm has optimised in both possible directions it tests a point extrapolated from the start and finish point of the two-dimensional search, that is, the points P1 and P3. The location of P4 is determined so that it mirrors P1 about P3. If a parabolic interpolation through the points P1, P3 and P4 has its peak beyond P3, towards P4, then it indicates some form of ridge rising up in the P1 → P4 direction. The algorithm therefore replaces one of the previous search directions with this new and better one. However, in this example no ridge is detected, so the algorithm performs another iteration over the original directions, finding points P5 and P6 in the process.

The linear independence of the search directions is necessary to guarantee that the whole search space can be examined. Any search using even one linearly dependent direction will find a false maximum within a reduced section of the search space. However, keeping the directions linearly independent while adapting to the best maximisation direction is difficult, and so the directions must either be reinitialised to some linearly independent set¾as implemented by Brent (1973), which is quite complex, or by heuristic means.

3.5 Implementation of Powell's Algorithm (Direction Set Methods in Multi-dimensions)

Because Powell's algorithm provides a framework for applying single dimensional maximisation without specifying any restrictions on the line maximisation method, it should be obvious that the 'best' method for the problem of interest should be used. In this case the single dimensional optimiser used is "Brent's method in one dimension" (Press *et. al*. 1992), which should provide rapid convergence where the function is well approximated by a parabola. This means that if the local curve approximates a parabola, then the best point can be found very quickly (in one step if the surface is truly parabolic). This algorithm was chosen because it has good performance and a small upper bound on the maximum number of steps, while using a relatively simple, non-problem specific, heuristic (Brent 1973).

There were several problems getting both Powell's and Brent's algorithm to work with the UDDER simulator. These all related to the assumption that the surface to be optimised is continuous and unbounded while in fact it is discrete, to a specific resolution, and bounded within user defined limits, both of which constraints are typically different for each variable. The problem with the discrete resolution of the search space is that the stopping conditions for various parts of Powell's algorithm are

based on a relative accuracy, in relation to the resolution of the computer's floating point accuracy. Changing the algorithm to be accurate to a specific (and arbitrary) accuracy of one decimal place was the solution used. This requires that all the variables used in the optimisation were scaled to provide that arbitrary precision. Bounds constraints were handled by imposing a linear penalty on the calculated fitness for any variable outside the bounds, as indicated in Fig. 6.

3.6 A hybrid optimisation algorithm

A hybrid algorithm was developed in an attempt to mix the searching ability of the GA with the speed of a hill-climber. This starts with a GA run which terminates when the rate of increase in the best fitness found falls below some arbitrary small value. This step utilises the GA's ability to find good areas in the search space quickly, as shown by the initial slope in the fitness graphs (Fig 8, discussed in Section 5). Once the GA has stopped, the best point is used as the starting point for an optimisation run by one of the hill-climbers. Because the DirSet algorithm generally outperformed the Simplex method, it was chosen for these experiments. The hill-climbing finish is used to overcome the difficulty that the GA has in climbing the last slope to the optimal point.

3.7 Caching Scheme

A typical GA optimisation run may require 1000's of simulations of a whole season farm performance, each taking around 4 sec on a 66MHz 486DX PC. To speed-up overall performance a cache memory (implemented as a hash table of linked lists) stores the input conditions, and output, of every simulation that is computed. When a new simulation is requested, a cached result can then be returned if it already exists. The cache is particularly effective in the final stages of the GA optimisations when almost all the requested simulations will have already been done, and total GA run times can be reduced by up to 80%.

4 Description of Experiments

Two experiments were undertaken to compare the performance of the various algorithms. Because the results of a single optimisation run can be quite dependant on the starting conditions, some form of statistical comparison is required. Hence each experiment involved performing a set of ten separate optimisations of the farm performance over a twelve month season for each algorithm investigated. Each of these optimisations commenced from a different, randomly-generated, starting point in the space of the optimisation variables. The performance of an algorithm was then characterised by the mean, standard deviation, and extreme values of the farm performance at the end of an optimisation run and the computation time needed to achieve that result. In all cases the measure of performance of the farm was the total milk-fat production over a twelve-month season. (Different measures may be more relevant to current farm practice, but such choices are unlikely to affect the concusions of this investigation).

The Powell's Direction-Set (hillclimber) algorithm was terminated when no improvement was found after testing all directions, and the directions tested have not been changed (with a maximum of three restarts). For the GA the condition was 100 generations with no improvement to the best point, with an (infrequently needed) over-riding cut-off of 1000 generations or 95% of the population converged to the same point. The transition from GA to hillclimber in the hybrid algorithm took place when the difference between the best individual of the generation and the mean of the previous generations (weighted towards the most recent) was less than 0.1%.

The first (and main) experiment considered a relatively large and complex problem with 15 optimisation variables, all of which were related to management of the farm's internal resources. ie it was representative of the type of optimisation that would be required in practice by a farmer or adviser for a realistic farm system. The second experiment considered a much simpler (6 variable) problem for which identification of the true optimum was much more likely for all algorithms. The main aim of this was to provide the basis of a comparison of the performance of the various algorithms on 'difficult' and 'relatively easy' problems

The optimisation variables in the large problem were stocking rate, the start dates of (pre-determined) calving and dry-off patterns, and the pasture allocation (expressed as a rotation length) in each of the twelve months of the year. These are representative of the main 'internal' variables that are under the farmer's control and, due to the subtelties of the animal-pasture interactions that are impacted by these

variables, the optimisation surface in this 15-dimensional space is undoubtedly highly structured.

For the small problem the stocking rate, calving and dry-off variables were retained while the 'difficult' pasture allocations were replaced by bought-in hay fed to dry cows in three periods. Since in the optimisation criterion used there was no penalty for the dollar cost of such feed, these variables would not have given rise to much structure in the optimisation surface (and in fact all the optimisation algorithms readily found the 'obvious' solution of making them as large as possible).

In the main (15 variable) experiment two variations on the GA were examined, and these were also used in the GA section of the hybrid algorithm. The first used a population size of fifty, a tournament size of three, and two-point crossover. The other, designated by *small*, used a population size of ten, a tournament size of two, and uniform crossover. The rational behind the different crossover methods used is described in Section 3.3.5 The larger tournament size with the population of fifty was used ensure sufficient selection pressure. A population sizing scheme based on the number of bits in the genome (Goldberg, 1989) suggests that a population of 116 should almost guarantee finding the optimum. However that would at least double the already long run times and, given the success of the hybrid algorithm, was not considered necessary.

The smaller (6 variable) problem used only a single GA setup because the the smaller population size of twenty was clearly adequate. A separate series of optimisations with a population of ten would not have produced significantly different results, and using a population size smaller than ten is likely to decrease the searching ability of the algorithm significantly (Reeves 1993). The other parameters used in this experiment were two-point crossover and tournament selection using a tournament size of two. The GA section of the hybrid also used these options.

5 Discussion

The results of the experiments described above are summarised in Tables 1 and 2 respectively, and in Figs. 7 and 8 the progress of each of the algorithms with computation time (expressed as the number of whole-season simulations of the farm) is shown for the main (15 variable) experiment. Table 3 illustrates the final values of the optimisation variables for the most important algorithms in the main experiment.

In both Table 1 and 2 the statistics shown for the hill-climbers (Simplex and DirSet) are based on only nine of the ten optimisations. The missing optimisation started from a point where all of the cows died, and hence the farm had a fitness of zero. Because neither hill-climber escaped this 'region of death', the final result of zero fitness skewed the average and standard deviations severely. So to provide a better indication of the true performance those optimisations were ignored. The other algorithms were not affected as much by such difficult starting points as they begin with a set of randomly chosen points (the genetic population), of which typically fewer than half give zero-fitness farms, and thus the problem is avoided.

The results from Table 1 show that while DirSet can produce very good results in a short period of time, it is inconsistent in the 'optimum' it finds. The opposite is true for GA, which is extremely slow, but can produce very good results indeed. From Table 2 it is clear that in this simpler problem the more robust searching capabilities of the GA and Hybrid algorithms give no advantage in finding the optimum (and a very major disadvantage in the time taken to do it) over a 'good' hillclimber (the Direction Set algorithm).

The relatively poor performance of the Simplex method (even in the small problem) is due partially to the uniformly small number of points tested, which significantly reduces the probability of finding good points. The other reason for its lack of success is due to the region of the search space examined, which is typically more restricted than the other algorithms. These flaws might be reduced by increasing the search area, which will in turn lead to longer convergence times, and hence more points tested. It should be remembered however that when UDDER was developed the performance of the "average PC" was such that only around 50 farm simulations per hour could be computed. Thus at that time the Simplex method was probably the only practical optimisation tool.

The performance of the hybrid algorithm, for either GA configuration, is equal or superior to all the other algorithms. Both provide consistently good results (as shown by the standard deviation, and the difference between the best and median optimisations) relatively quickly. The small population hybrid is

probably the best mixture of reliability and speed, especially given that two optimisations can be performed.

The graphs in Figs. 7 and 8 show the best, median and worst optimisations, as given in Table 1, as a function of number of simulations (or algorithm execution time). The value shown is the best result found up to that point, where this is given for every simulation of the hill-climbers and only once per generation of the GAs (10 or 50 simulations — the population size). The generational depiction of the GAs is because the simulations within a generation have no inter-relation. A depiction of exactly when, within a generation, a value was reached has no significance, unlike the hill-climbers where each and every simulation is related in some form to those directly before it. Note in Figs. 7 and 8 that the labels 'best', 'median' and 'worst' apply to result obtained at the (arbitrary) termination point of the experimental run; as is indicated by the crossings of the lines in these figures, the run which produced the best end result usually did not give the best result at all the earlier times.

Figs. 7 and 8 highlight the differing behaviour of the algorithms. The GAs and Hybrids appear to start with consistently better points than the hill-climbers, but that is because only the best in the starting population is shown. The final position of the median run for each of the GA and Hybrid optimisations shows that the optimisations tend towards the best. For the Hybrid that *skew* is such that the final points of the best and median runs are indistinguishable on the graphs. The hill-climbers show a much more balanced range of final points.

The large increases in fitness towards the end of the small population GA show the speed with which a good individual can influence the entire population. A similar looking increase shown by the median run of the large population GA is, because of the log scale, actually an order of magnitude slower.

One important point about the behaviour of the Hybrids that is not shown in either table or graph is the division of optimisation between the GA and DirSet sections. For the large population Hybrid the GA runs for a long time, as it continually finds improvements. DirSet occurs only in the very last stages of optimisation. For the small population Hybrid, however, the workload is the other way around; the majority of the optimisation being performed by the DirSet algorithm, with only the initial one to two hundred simulations (ten to twenty generations) used in the GA stage. This works because the DirSet stage is most effective from a good starting point and thus the GA need only obtain a very rough estimate of the good region. The small population sized GA does that quite efficiently. In fact, the number of simulations used in the DirSet stage for the small population hybrid is only approximately 30% more than for the large population. Yet the small population hybrid still finishes in approximately half the time of the large population hybrid (Table 1). The danger in using a small population is that there is a greater chance that the good region will be missed entirely, forcing the hill-climber to work much harder. This results in the lower average and larger standard deviation shown by the small population hybrid.

The fitness values found in this series of experiments are, at best, a small improvement over the 'standard farm' (with a fitness of 53232), which is already good. The largest improvement on that standard farm's strategy was found by the large population GA, with an increase of 989 kg milk-fat production over the year's milking. This correlates to an improvement of about 2% over an already good farm. That is, however, an increase that occurs without additional monetary input (ignoring any cost involved in increasing stocking rate); a not insignificant result.

It is worth noting that the *average* GA still finds a strategy approximately equivalent to the standard farm after testing only ≈ 15000 of the $\approx 10^{33}$ possible points. Neither hill-climber was, on average, able to match even the standard farm's performance (and even the best Simplex run could not equal it).

The variables in Table 3 describe the standard farm, and the results after the best optimisations by Powell's direction set method, genetic algorithm, and hybrid algorithm. From this table it can be seen that the improvement is gained by more efficient pasture production: the shorter rotation lengths in the period of fastest growth (Sept to Nov) essentially provide more pasture to the cows without reducing the grass to a level where the regrowth is compromised. Better utilisation is achieved through the slightly increased stocking rate and length of milking season (as indicated by the calving to dryoff interval). These results make good sense in the light of current understanding of pasture management.

6 Conclusions

This work has investigated the application several existing optimisation algorithms to a computer simulation model of a pastoral dairy farm¾a complex, high-dimensional system of practical importance. Powell's direction set method and the genetic algorithms both provide a more powerful optimisation tool than the built-in downhill simplex method in UDDER. However, they still have several weaknesses when applied to complex problems. The direction set method has a tendency to find sub-optimal peaks in the fitness surface, or to become lost in the large 'flat' areas that occur for some variable sets. These local quirks in the search space are problematic for any hill-climber due to their reliance on the local gradient information. Genetic algorithms, on the other hand, work best in the early stages of the optimisation, narrowing down the search to the good regions quickly. Their eventual climb towards the optimal point is, however, very slow in comparison to the other algorithms. In essence the hill-climbing and genetic algorithms are the extremes of optimisation, one quick yet error prone, the other robust yet slow to finish.

A solution to those problems was attempted by constructing a hybrid algorithm which combines the strengths and minimises the weaknesses of its components. The implementation developed here is the most simple approach: a coarse-grained algorithm which runs first the genetic algorithm for its initial good region finding ability, followed by the hill-climber to quickly find the optimum in that region. Other possibilities that were not examined include a fine-grained Lamarckian evolutionary model, where the genetic algorithm's individuals are further transformed each generation by a few steps of a hill-climber.

From the experiments with these algorithms we conclude:

i) The hybrid algorithm performance, in terms of the optimisation time, falls between the extremes of GA and hill-climber, while the final optimisation result remains good in almost all cases.

ii) The good final result is dependent on the ability of the hill-climber to do well when starting from a moderately good point. This allows the use of a faster but less accurate search by ta small-population GA.

Without examining more than the single hybrid algorithm, it is still possible to conclude that this is no less robust than the genetic algorithm for finding good solutions, while performing only slightly more slowly than a single run of the direction set method by itself. This approach has therefore provided a useful improvement on the individual algorithms. This result is likely to hold true for other hybrid algorithms, assuming that they can be constructed so that the strengths of one offset the weaknesses of the others.

While the results obtained by the hybrid algorithm are almost always good, it must be recognised that they are never guaranteed to be the true optimum. This is true of any algorithm that does not perform an exhaustive search of the entire space, which in practice means any algorithm. For a continuous surface such an exhaustive search is, of course, impossible: even with the discrete space of the simulation an exhaustive search of the 15 variable optimisation would require more than 10^{26} years, and thus is (and always will be) computationally intractable.

With respect to the dairy farming aspect of this work, the simulation and optimisation of the farm allows literally thousands of years worth of empirical manipulation of actual farm management to be performed in a matter of hours. This level of experimentation should, at the very least, provide a sense of what changes will improve any specific farm, and has the advantage that unlike any real farm experimentation the comparisons are made under controlled conditions. The fact that the simulation is not perfect, as no simulation can be, should not degrade the general results obtained by such simulations. This assumption is implicitly required for the optimisation results to have any real world applicability. Obviously, any improvements in the accuracy of representation of the underlying simulation model will translate into more valuable output from the optimiser.

In conclusion, the techniques examined are applicable not only to this specific problem, or to a wider range of dairy farming strategies, but to others involving the simulation of real world situations. Future work in the area of optimisation might include additional comparisons of newer genetic algorithm techniques and other search strategies. Certainly a determination of 'optimal' GA parameters for specific problems, and from those good heuristics for parameter choice, would benefit the GA optimisations. The other types of hybrid algorithm, along with hybrids of different algorithms, should be tested, and there is

no reason that three or more algorithms can not be successfully used together in this way.

Acknowledgements

This work was supported by the New Zealand Foundation for Research, Science and Technology.

References

Agricultural Research Council (1980) Nutrient requirements of ruminant livestock. Technical review by an Agricultural Research Council working party. Commonwealth Agricultural Bureau, Farnham Royal, Slough, England.

Brent, R.P. (1973) Algorithms for minimization without derivatives. Prentice-Hall, Englewood Cliffs, New Jersey.

Darwin, C. (1899) The Origin of Species, 6th edition. John Murray, London.

Dawkins, R (1996) Climbing Mount Improbable, Viking Press, London.

De Jong, K.A. & Spears, W.M. (1990) An analysis of the interacting roles of population size and crossover in genetic algorithms. In: H.P. Schwefel and R. Männer (Editors), Parallel Problem Solving from Nature, Springer-Verlag, New York, pp. 38-47

Garey, M.R. & Johnson, D.S. (1979) Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, San Francisco.

Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts.

Hart, R.P.T. (1996) Experimental comparisons of optimisation techniques - computer optimisation of dairy farm management. M.Phil Thesis, University of Waikato, Hamilton, New Zealand.

Larcombe, M.T. (1989) The effects of manipulating reproduction on the productivity and profitability of dairy herds which graze pasture. D.Phil. Thesis, University of Melbourne, Australia.

Nelder, J.A. & Mead, R. (1965) Computer Journal, 7: 308-313.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. & Flannery, B.P. (1992) Numerical Recipes in C, 2nd Edition. Cambridge University Press, Cambridge, England.

Reeves, C.R. (1993) Using Genetic Algorithms with Small Populations. In: S. Forrest (Editor), Proc. 5th International Conf. on Genetic Algorithms. Morgan Kaufmann Publishers, Inc., San Francisco, pp. 92-99.

Appendix A: Dairy Farming Glossary

Agistment: The practice of paying other farms money to feed your stock on their land.

Dry Cow: Any cows not currently producing milk

Calf: A cow under one year old.

Calving Pattern: A description of the number of cows that calve in each time period.

Condition (score): A measure of the fat reserves carried by an animal.

Dryoff Pattern: A description of the number of cows in each time period that stop producing milk.

Heifer: A cow in her first lactation.

Milker: A lactating cow.

Rotation: The movement of cows about the farm. Each paddock is used for a certain time, then the cows are moved to the next. The total time taken for the cows to cycle through all available paddocks is the rotation time.

Supplementary Feeding: Dispensing additional feed (for example, hay or silage) at times when pasture growth is low.

Topping: Process of cutting the top section of grass stalks to delay seed-head formation.

Figure Captions

Fig 1 An illustrative fitness landscape for two variables in the UDDER farm simulation.

Fig 2 Single point crossover.

Fig 3 Pseudo-code for the Simple Genetic Algorithm (SGA).

Fig 4 Two point crossover.

Fig 5 Several iterations of Powell's Direction Set algorithm on a simple surface.

Fig 6 Fitness cirve for one variable showing fitness penalty effects.

Fig 7 Optimisation results for the Simplex and Direction-Set algorithms. The different lines are the experiments which resulted in the best, median and worst final results.

Fig 8 Optimisation results for the GA and Hybrid algorithms (small refers to GA population size). The different lines are the experiments which resulted in the best, median and worst final results.

Table 1

Stocking, Calving and Dryoff start, Calving to Dryoff interval, All rotation (15 variables)

		Average	Std. Dev.	Best	Median	Worst
Simplex	Result Simulations	42787 301	2900 86	46970 307	42571 341	37791 314
DirSet	Result Simulations	51007 797	3104 252	54148 849	50568 698	46669 530
GA	Result Simulations	53125 15210	1612 3799	54221 15550	54043 16700	49710 11500
GA (small)	Result Simulations	51322 4791	2744 1284	54041 4580	52219 4700	46708 2740
Hybrid	Result Simulations	53425 2163	980 310	54121 1930	54012 1960	51307 2129
Hybrid (small)	Result Simulations	52283 1069	2736 280	54145 1379	53890 1155	46705 474

Table 2

Stocking, Calving and Dryoff start, Calving to Dryoff interval, Hay fed to Dry cows (6 variables)

		Average	Std. Dev.	Best	Median	Worst
Simplex	Result	50824	4113	54715	54096	45898
	Simulations	170	35	163	151	113
DirSet	Result	55049.7	0.5	55050	55050	55049
	Simulations	306	102	152	279	504
GA	Result	52129	2172	54752	52365	50062
	Simulations	3126	1528	4680	2400	2360
Hybrid	Result	55049.8	0.4	55050	55050	55049
	Simulations	588	95	467	599	668

Table 3

Original and optimised farm conditions from a 15 variable problem

	original farm settings	DirSet	GA	Hybrid
Stocking Rate	3.1375	3.1625	3.34	3.1625
Calving and Dryoff Pattern Start	23	23	23	23
Calving to Dryoff Interval	31	33	34	32
Rotation in Jan	20	12	12	12
Rotation in Feb	24	14	12	14
Rotation in Mar	28	19	15	17
Rotation in Apr	30	23	18	23
Rotation in May	30	23	35	23
Rotation in Jun	30	20	23	20
Rotation in Jul	20	181	141	82
Rotation in Aug	20	11	161	11
Rotation in Sep	20	11	12	11
Rotation in Oct	20	11	11	11
Rotation in Nov	20	11	15	11
Rotation in Dec	20	21	14	21
Fitness	53232	54148	54221	54145

Fig. 1 An illustrative fitness landscape for two variables in the UDDER farm simulator

Fig. 2 Single point crossover

```
create initial random population [size N]
while (stopping condition not met)
        for (every individual in current population)
                calculate that individual's fitness
        end for
        for (every individual in new population [also size N])
                choose (two parents from current population, based on their fitness)
                new individual = crossover (parents)
                mutate (new individual)
        end for
        current population = new population
end while
```

Fig. 3 Pseudo-code for the Simple Genetic Algorithm (SGA).

Fig. 4 Two point crossover

Fig. 5 Several iterations of Powell's algorithm on a simple surface

Fig. 6 Fitness curve for one variable showing fitness penalty effects

Fig. 7 Optimisation results for the Simplex and Direction-Set algorithms. The different lines are the experiments which resulted in the best, median and worst final results.

Fig. 8 Optimisation results for the GA and Hybrid algorithms (small refers to GA population size). The different lines are the experiments which resulted in the best, median and worst final results.