Scheduling chicken catching – An investigation into the success of a genetic algorithm on a real-world scheduling problem

Emma Hart, Peter Ross and Jeremy A.D. Nelson
Division of Informatics, Artificial Intelligence Applications Institute, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, Scotland, UK

E-mail: {emmah;peter;jeremyn}@dai.ed.ac.uk

Genetic Algorithms (GAs) are a class of evolutionary algorithms that have been successfully applied to scheduling problems, in particular job-shop and flow-shop type problems where a number of theoretical benchmarks exist. This work applies a genetic algorithm to a real-world, heavily constrained scheduling problem of a local chicken factory, where there is no benchmark solution, but real-life needs to produce sensible and adaptable schedules in a short space of time. The results show that the GA can successfully produce daily schedules in minutes, similar to those currently produced by hand by a single expert in several days, and furthermore improve certain aspects of the current schedules. We explore the success of using a GA to evolve a *strategy* for producing a solution, rather than evolving the solution itself, and find that this method provides the most flexible approach. This method can produce robust schedules for all the cases presented to it. The algorithm itself is a compromise between an indirect and direct representation. We conclude with a discussion on the suitability of the genetic algorithm as an approach to this type of problem.

Keywords: genetic algorithm, real-world scheduling, evolving heuristic strategies

1. Introduction

Since the earliest work by Davis [3] in 1985, there has been an increasing amount of literature on the use of genetic algorithms for solving scheduling problems, for example see [19]. However, the majority of this work addresses benchmark or artificial problems, where there is a known solution to a defined problem. A much smaller proportion of work is directed towards attacking real-life problems, which often turn out to be much more difficult than a study of the benchmarks would suggest. Some promising work on various real-life problems, however, see [10,15], suggest that the genetic algorithm approach is worthwhile investigating.

Tackling real-life issues introduces many additional factors and constraints into a problem, aside from the usual constraints normally associated with scheduling problems [13]. Whilst many of these constraints just relate to practical issues concerning the particular problem, others cannot be so easily defined and can also be somewhat subjective. A human scheduler may do an excellent job of producing working schedules, but the methodology they follow varies from day to day, and the decision to use a particular method comes only as a result of years of experience. Often, the precise approach taken cannot easily be formulated. Surprisingly little is known about the way humans plan and schedule, and is the subject of ongoing research [11].

In the case discussed, the company is not concerned with finding *optimal* schedules that minimise makespan or resources, but in having an automated system that produces practical working schedules. Hence, the problem is one of *constraint satisfaction*, to find schedules that minimise the number of violations of a set of constraints, yet remain practical in the sense that they are compatible with the current working practice of the company. An overriding criterion was that the automated schedules produced should be similar to those currently produced by hand at the company. This was seen as necessary in order for the company management to have confidence in the schedules and raises an interesting sociological issue. Defining a suitable set of constraints in order to try to emulate the thought process of the current scheduler proved essential to the success of this study and involved considerable discussion and knowledge acquisition. The problem is discussed in detail in the next section.

2. Problem description

The problem relates to a local firm that catches around 1.3 million chickens a week, and kills them at two factories which are situated at Newbridge, close to Edinburgh, and at Coupar Angus, approximately 70 miles away. The birds are caught live by teams of catching *squads* at farms geographically spread across Scotland and Northern England, and are delivered to the factories by teams of lorries and drivers.

The scheduling task is a daily task to schedule catching squads and lorries to deliver a set of *orders* in such a way that the factories are continually supplied with birds throughout the day. An *order* is defined as a number of *modules* of birds of a particular *type*, that must be collected from a farm and delivered to a factory. Each day, up to 15 orders may need to be scheduled, each consisting of up to 8 lorry loads. A factory must not become idle at any time – however, strict regulations govern the maximum period a lorry load of birds can be kept waiting outside the factory, hence loads should only arrive at the same rate as the factories can process them.

The process must take into account a fixed resource of squads, lorries and drivers, and ensure that the contractual requirements of the squads and lorry drivers are met, as well as producing sensible working schedules. Currently, each day is scheduled separately, although there are also various constraints that must be satisfied when looking at a week as a whole.

Some of the constraints are expanded below:

2.1. Squads

A *squad* is based at one of the factories and works either part or full time. It rotates between three shifts, classified as *early*, *floating*, or *late*, depending on the time of day the shift begins. There are contractual requirements specifying the minimum and maximum number of modules a squad can catch in one day, and also in one week, and the maximum length of the shifts they can work. Some farms have their own catching squads, and there are restrictions on the order in which squads can visit certain farms due to diseases existing at some farms.

A squad leaves its base factory at the start of the day and is assigned to catch N_i loads at each of n farms, F_i. After catching N_i loads at farm F_i, it travels directly to farm F_{i+1} without a break, and eventually returns to its base factory only at the end of its shift. The time taken to catch each load is directly proportional to the size of the load and is independent of bird type.

2.2. Lorries and drivers

A lorry must be scheduled to meet each of the N_i loads at each farm as they are caught, and deliver them to a factory. Each factory runs its own fleet of lorries, and the lorries are available in two different sizes. Certain types of birds require a large lorry for delivery, whereas others can fit in either size.

Whilst a lorry may deliver loads to either of the factories during the day, it must deliver its last load to its base factory in the evening.

Drivers must also be assigned to each lorry trip – a lorry driver can work a maximum of a 13-hour shift, of which only 9 hours can actually be driving. If necessary, contract drivers can be hired in to do any extra work, though of course it is preferable to avoid this.

A diagram representing the work pattern for two squads is illustrated in figure 1, showing each squad travelling around a circuit of farms, and a lorry collecting each load as it is caught.

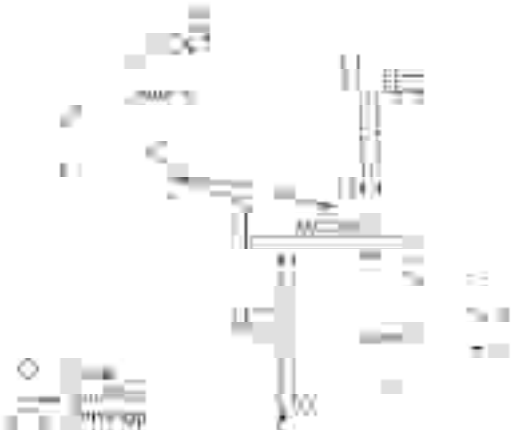

Figure 1. Illustration of squad and lorry trips for two squads collecting work for a single factory.

2.3. Factories

As noted previously, each factory must be continuously supplied with birds, with ideally an even time lag between load arrivals. Birds must not be kept waiting outside the factory for long periods of time, especially in summer. A factory may kill birds of several different types during the day, each of which has a different processing time. It may have a preference for the order in which it kills certain bird types, and hence the order in which they arrive.

3. Previous work

Several evolutionary approaches to this problem have been investigated. Whilst only the most successful is reported in detail in this paper, we provide a brief overview of the previous work in order to highlight some of the difficulties found when tackling this chicken scheduling problem.

The first two models attempted to produce squad, lorry and factory schedules simultaneously. A *direct* model [2] was initially investigated, in which the complete schedule of squads, lorries and factory arrivals was represented on one chromosome. In contrast, the second method took an *indirect* approach [1]. This used a representation in which the chromosome encoded a permutation of orders that were fed into a schedule builder. The schedule builder consisted of a set of instructions for assigning the order to a squad, allocating a lorry to deliver the order, and scheduling the arrival time of the order at the factory. The critical component of the system turned out to be the schedule builder, rather than the genetic algorithm. A large amount of specific knowledge needed to be encapsulated, and the resulting rule-based schedule builder provided a rather inflexible approach to constructing schedules. In both of these models, juggling the combined constraints of the transport fleet, squads and factories resulted in an extremely difficult scheduling task, in which modifying one part of the schedule tended to produce an unpredictable or large change in another part. Subsequent studies hence followed a more modular approach in which the problem was broken down into two distinct parts.

4. Divide and conquer

Tackling the whole problem of scheduling squads, lorries and arrivals for both factories at once introduces unnecessary complexity and extra constraints. Extensive discussion with the human scheduler showed that the scheduling problem naturally breaks down into two separate stages. In stage I, the set of orders for each day is broken down into tasks, where a task is a unit of work that can be performed by a squad. Each task is then assigned to a squad. Stage II deals with scheduling the arrival of these tasks to each factory.

There are restrictions and preferences governing the combination of tasks into squad work loads – these include a preference that the farms visited by a squad are

located in the same geographical area, and restrictions on the total amount of work a squad can do per day. All tasks at a farm are always of the same type, and hence there is no preference for the order in which they are carried out at a farm. There is, however, a preference for the order in which the set of farms assigned to a squad is visited by that squad, as this has a direct effect on the order of arrivals at the factory.

Once a satisfactory assignment of work to squads has been found, the stage II process schedules the arrival of each task to the factory. A squad must work continuously without any breaks from the moment it begins its shift (where "working" implies either catching birds or travelling between farms). A lorry is scheduled to arrive at a farm at precisely the same moment as the squad, and modules of chickens are loaded directly onto each lorry by the squad, hence the lorry is ready to leave the farm at the same moment the squad finishes catching chickens. Thus, timing factory arrivals reduces to scheduling a *start-time* for every squad. Given a sequence of farms visited by a squad, specifying a time at which work begins at the first farm in the sequence is sufficient to determine the arrival time of every load the squad catches. This step makes the assumption at this stage that there are sufficient lorries available to meet each load as it is caught – we show later that this is in fact a valid assumption.

The problem can be simplified further by treating each factory as an individual scheduling problem, and allocating work and scheduling arrivals for each factory separately.

Thus, two genetic algorithms are implemented. The first, an *assignment GA*, described in detail in section 6, produces an assignment of squads to work. The second, the *timing GA*, described in section 8, allocates a start time to each squad, such that the arrival times at the factory of each load produce a balanced schedule.

5. Assigning work to squads

The purpose of implementing an assignment GA is to distribute the collection of a day's orders amongst the available squads. Each order is broken down into a set of tasks, T, where a task $T_i(m, f)$ represents m modules of birds to be collected from farm f. The aim is to produce an assignment of tasks to squads that satisfies all constraints. This can be viewed as an analogy to a bin-packing problem – work must be placed into bins, where each bin is a squad, so that the bins do not overflow, and items in the bin do not conflict. Several people have reported the successful application of a GA to this type of problem, for example [7,9].

Early attempts at solving the module-squad assignment problem investigated a very simple direct approach. Each chromosome consisted of N genes, where N is the total number of tasks created by breaking down each order. The most natural task size to work with is that of a complete lorry load of modules. A standard sized lorry can hold 22 modules. An order of M modules thus breaks down into $\lfloor (M-1)/22 \rfloor + 1$ tasks. (The final task could consist of a partial lorry load.) However, a wide range of other task sizes was also investigated. Each gene has a value that denotes the squad

that must perform the task corresponding to the gene and is initialised at random with one of the squads that are available that day at the factory in question.

The approach proved very sensitive to the number of tasks *N* represented by the genome: Splitting the farm orders into very small tasks ensures that if there *is* a schedule that satisfies all constraints, then it must exist in the search space. However, the search space is so large that finding this schedule is extremely difficult, and although the GA found an acceptable solution on occasion, it often converged to a local optimum. With small task sizes, a large number of solutions represented impractical schedules, where squads collected small number of birds from several farms, and resolving these conflicts hindered the search for good solutions.

Decreasing the granularity of the task size, so that each task represents a larger unit of work, dramatically reduces the size of the search space. However, this is counter-balanced by the risk of excluding the optimal solution from the search space altogether if the unit size is too large. Finding an acceptable trade-off between task size and search-space size is difficult – careful examination of a large number of the schedules produced by hand at the factory showed that it was essential to include *some* tasks of very small size in order to produce the best schedules, but also that several farm *orders* could be scheduled as single tasks. Clearly, some compromise between the two extremes is needed.

6. Evolving an assignment strategy

The early work highlighted two main problems. Firstly, using a direct approach to the assignment problem as described above, it was difficult to define a search space that would include the optimum solution but was small enough to search rapidly. Secondly, we found that using a permutation + schedule builder approach as described in section 3, the schedule builder was too rigid and too inflexible to cope with the variation inherent in the problems from day to day. We present a GA, GA_{S1}, that circumvents these problems by switching its focus from trying to evolve a *solution* to evolving a *strategy* for tackling each day's scheduling, and then apply that strategy to produce a schedule or solution.

Given a sequence of orders, the job of the strategy is to decide how to break each order down into a set of suitably sized tasks for scheduling, and then assign each of those tasks to a squad using a heuristic. Each chromosome encodes the sequence in which orders are to be considered, a set of heuristics for breaking each order down into tasks and a further set of heuristics which determine how the tasks are distributed amongst the squads. The GA must evolve the best set of heuristics to use in each individual case. The quality of a given strategy is determined by evaluating the schedule produced from it against a set of constraints defined in a fitness function. This function, and the set of constraints used, is described in section 6.5.

The result is a separate schedule builder being built for every schedule created, each using a mixture of tasks sizes and assignment heuristics highly specific to each order.

6.1. Representation and methodology

Work destined for each of the two factories is treated separately. Both factories are represented on the *same* chromosome, and the strategies for scheduling their work evolve simultaneously. A squad may be assigned work that is destined for either factory, regardless of where it is based. By evolving the factory schedules simultaneously it is possible, for instance, for an under-worked "Coupar" squad to alleviate an overworked "Newbridge" squad or vice versa, with the result that the load balance across squads becomes more even.

The structure shown in figure 2 represents the strategy to be used for building the assignment of work to squads for a single factory. Each chromosome consists of two such structures, one for each factory. The strategy is decoded in a manner described

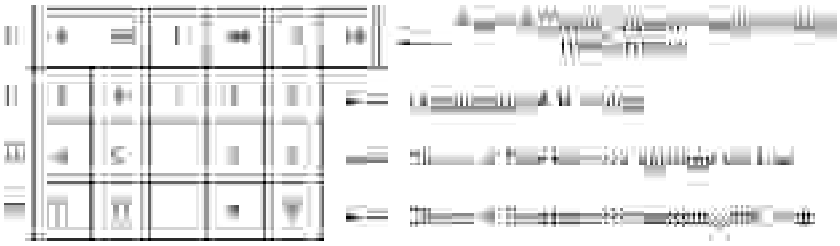

Figure 2. The chromosome representation used in GA_{S1}.

in the following sections, and the resulting assignment of squads is evaluated against a set of constraints for each factory separately. The overall fitness of the chromosome is a linear combination of the fitness of assignment at each factory.

After a fixed number of evaluations, the assignment produced as a result of the best strategy found is fed to the timing GA, GA_{Timer}, which outputs the lorry and arrival schedules and is described in section 8.

6.2. Section I – Incorporating domain knowledge

There are certain criteria which must be satisfied by every schedule produced. For example, the Coupar Angus factory must always process a large load of birds of type "Roaster" when it opens, and close after processing a load of birds of type "Medium". We can cut down the search space by *guaranteeing* that these criteria will be met in *every solution*. In order to preserve flexibility, we allow GA_{S1} to evolve an answer to exactly *how* these criteria will be met. As illustrated in figure 2, section I, pairs of genes specify an *order* and *quantity* for satisfying each fixed criterion, with one pair for every fixed criterion.

Thus, the Coupar Angus requirement for a large load of Roaster birds to open the factory is met by the first (*order*, *quantity*) pair shown in the example in figure 2

– (4, 96). This indicates that 96 modules from order 4 should be used to satisfy this requirement. In this way, we ensure that the constraint will be met, but there is considerable scope for specifying how it is met, in that both the farm from which the birds are taken, and the precise interpretation of "large" are specified by the chromosome. The approach incurs a small overhead in that a *repair operator* must be applied to each chromosome to ensure that the pair (*order*, *modules*) is feasible, i.e. the number of modules specified is not greater than that contained in the order.

6.3. Section II – Sequencing the orders

This section of the chromosome contains a permutation of the N orders that must arrive at the factory that day, and indicates the sequence in which they should be considered by the strategy defined by sections III and IV of the chromosome for breaking down into tasks and assigning to squads.

6.4. Sections III and IV – Splitting and assigning an order

In section III of the chromosome, N genes, denoted S_N, specify how each of the orders should be broken down into schedulable tasks. A further N genes in section IV, denoted A_N, specify how the tasks should be assigned.

The ith order in the permutation is split according to the heuristic indicated by gene S_i and then each task assigned according to the heuristic indicated by gene A_i. Tables 1 and 2 list the possible heuristics that can be used in each case.

6.5. Fitness

Each assignment must be evaluated in terms of its ability to satisfy a set of constraints. The constraints used are defined in table 3. The question of how to deal with constraints and assign some numerical fitness value to a chromosome for the purpose of selection has been widely discussed in the GA literature. The reader is referred to work in [6,14,20] for at least three different types of approach. Here, we adopt a penalty function approach, where each violation of a constraint incurs a penalty – the penalty is weighted according to the importance of the constraint. The overall penalty of any assignment is given by

$$penalty = \sum_{i=0}^{i=N_{Cons}} w_i n_i, \qquad (1)$$

where i denotes a constraint, n is the number of instances in which the constraint is violated, w is the weight attached to the constraint, and the sum is over all constraints N_{Cons}. The fitness assigned to a chromosome is given by $1/(1 + penalty)$.

Choosing the correct weight for each penalty can be a non-trivial task, and is discussed in [4] and [17]. In this case, finding acceptable penalty values turned out to

Table 1

Heuristics for splitting an order.

Heuristic	Description
A	If farm order ≤ 110, use the whole order as single task
B	Split into single loads of 22 (but split first load into 10 + 12)
C	Split into tasks of size 44
D	1 task of size 66, split rest into single loads
E	Split into tasks of size 66
F	1 task of size 88, split rest into single loads
G	Split into tasks of size 88
H	1 task size 110, split rest into single loads
I	If order = 110, split into 56 and 54
J	If order = 110, split into 66 and 44

Table 2

Heuristics for assignment.

Heuristic	Description
A	Assign each part-load to the first squad found that satisfies all the constraints
B	Try to assign the load to a squad that is already doing this type of work (taking into account constraints)
C	Assign the load to an unused squad (if possible)
D	Assign the load to a squad already working (if possible)
E	Assign the load to a squad already going to this farm
F	Assign the load to a squad already going to the same *complex*
G	Assign the loads randomly

Table 3

Description of constraints used, and their associated weights.

Penalty	Description	Weight
Overload	each squad *s* can catch a maximum of Max_s modules per day	10
Underload	each squad *s* must collect a minimum of Min_s modules per day	1
BothSides	a squad should not visit farms located North and South of the Forth Estuary in the same day	3
Total farms	the total number of farms visited by a squad in a single day should be minimised	1
Small load	A squad must collect at least a complete lorry load from each farm (or complex) it visits	10

be straightforward, using an empirical approach. The values used are shown in table 3. Note that in this case, the weights simply represent values that caused acceptable solutions to be found, and do not necessarily reflect the importance of each constraint. For example, the *underload* constraint has a very low weight, despite being of considerable importance – this is due to the fact that the assignments found by the GA rarely incurred any underload violations.

7. Results and discussion

Experiments were performed using two real data sets provided by the factory, representing two complete weeks. One set was considered "straightforward" to schedule by the human expert, and the other "difficult".

7.1. GA parameters

All experiments reported were performed using a population of size 100 and a steady-state reproduction strategy [22]. For sections I, III, and IV of the chromosome, two-point crossover was applied. A position-based crossover operator, PMX [12], was applied to the permutation of orders. Mutation was applied with a probability 0.02 to each gene in each of sections I, III and IV – an order-based mutation operator that picks two loci at random and exchanges their alleles was applied to the permutation in section II. All experiments were run for 10000 generations.

7.2. Results

GA_{S1} produces good, acceptable schedules reliably for every day it has been tested on. In every trial, every constraint is satisfied – the final penalty arises *only* from the unavoidable fact that at least one squad must visit more than one farm. (Hence, a penalty of 0, i.e. fitness 1.0, is impossible to achieve.) Furthermore, when a squad is assigned to visit more than one farm, the set of farms visited is always deemed to be geographically "practical", in the sense that the set of farms visited is entirely located on either the North or South side of the Forth Estuary, and also practical in the sense that at least a whole lorry load of birds is collected from each farm or complex visited.

The results in table 4 show the minimum, average, and best fitness achieved in 20 trials with each data set.

7.3. Variation in strategy for splitting loads

Analysis of the best strategy evolved over 20 trials for each day showed that the evolved strategy for splitting the loads varies quite considerably with day tested. Choice of strategy used is clearly essential to the evolution of a good solution. The

Table 4

Results of experiments using GA_{S1}, averaged over 20 trials.

Day	Min fitness	Average fitness	Max fitness
Monday 10/2	0.200	0.200	0.20
Tuesday 11/2	0.200	0.248	0.250
Wednesday 12/2	0.500	0.500	0.500
Thursday 13/2	0.500	0.500	0.500
Monday 21/4	0.125	0.146	0.167
Tuesday 22/4	0.25	0.321	0.333
Wednesday 23/4	0.111	0.115	0.143
Thursday 24/4	0.091	0.118	0.143

Table 5

Frequency of application of splitting heuristic (percent).

Day	% frequency of heuristic being applied to split an order									
	A	B	C	D	E	F	G	H	I	J
Monday 10/2	83.6	2.9	10.7	1.4	0.7	0	0.7	0	0	0
Tuesday 11/2	77.5	5.8	1.7	0.8	14.2	0	0	0	0	0
Wednesday 12/2	43.75	6.25	37.5	0	0	0	10	0	1.25	1.25
Thursday 13/2	35	3.75	42.5	1.25	8.75	1.25	2.5	0	2.5	2.5
Monday 21/4	52.0	10.0	19.0	1.0	8.0	0	0	10.0	0	0
Tuesday 22/4	79.2	11.6	9.2	0	0	0	0	0	0	0
Wednesday 23/4	47.0	29.0	14.0	0	0	0	6.0	3.0	1.0	0
Thursday 24/4	43.0	33.0	8.0	3.0	9.0	1.0	0	2.0	1.0	0

distribution of frequencies of using a heuristic is obviously not random in any case considered.

The percentage frequency of application of each splitting heuristic, averaged over 20 trials, for loads arriving at Coupar Angus only, is shown in table 5. The reference to the heuristic refers to table 1.

In general, the most commonly used heuristic is “treat all of the orders as a single task” (heuristic A). In the first data set, this heuristic is almost always the most favoured one; however, for Wednesday and Thursday the heuristic that splits orders into chunks ofsize 44 is almost equally favourable (heuristic C). Notice that these two cases are the ones that also produce the highest fitness value – the same fitness is produced consistently in every trial, which suggests that perhaps the problem is fairly straightforward.

In the second data set, whilst the "1 task" heuristic A is again most popular, we notice a marked increase in the frequency of application of the "split into single loads" heuristic B, especially in the case of Wednesday 23/4 and Thursday 24/4. Interestingly, these two cases produce the lowest overall fitness, suggesting that they are difficult to solve. This set of data contains several orders that are too large to be assigned to a single squad, and hence must be split somehow – we see varying options for performing this split being chosen depending on the day, but the general trend seems to be towards performing a fine-grained split.

7.4. Variation in strategy for assigning loads

Table 6 shows the frequency with which each assignment heuristic was used, over 20 repeated trials. The heuristics themselves are defined in table 2. These results show a less clear picture emerging – the frequencies are much more evenly distributed and in some cases do not deviate significantly from the 14 frequencies which would

Table 6

Frequency of application of assignment heuristic (percent).

Day	% frequency of heuristic being applied to assign an order						
	A	B	C	D	E	F	G
Monday 10/2	15	19.3	12.1	15	14.3	12.9	11.4
Tuesday 11/2	8.3	14.2	12.5	12.5	12.5	28.3	11.7
Wednesday 12/2	16.25	16.25	10	12.5	17.5	16.25	11.25
Thursday 13/2	21.25	16.25	3.75	25	12.5	10	11.25
Monday 21/4	8.0	22.0	15.0	9.0	23.0	16.0	7.0
Tuesday 22/4	14.15	17.5	17.5	11.7	12.5	12.5	14.15
Wednesday 23/4	17.0	21.0	8.0	14.0	18.0	11.0	11.0
Thursday 24/4	26.0	16.0	10.0	15.0	11.0	10.0	12.0

be expected if the choice was completely random. Only in two cases – Tuesday 11/2 and Thursday 24/4 – do we see an obvious preference for one heuristic. These heuristics are "choose a squad already going to this complex" (F), and "choose the first squad found that satisfies all constraints" (A), respectively. Note that heuristic F cannot be applied to the first order assigned.

Overall, we can conclude that the major factor in the success of this GA is the flexibility achieved by allowing the GA to evolve a method of splitting the loads that is suitable to an individual case.

7.5. *Effort required to fine tune* GA_{S1}

GA_{S1} proved remarkably robust and needed very little parameter tuning in order to produce satisfactory results. Finding the correct operators and settings to use in a genetic algorithm can be something of a black art, and despite a considerable amount of research, e.g. [18], often just demands a straightforward empirical investigation.

Table 7 shows the effect of varying the population size on the quality of the solution obtained for the test case Tuesday 22/4, averaged over 20 trials. This case was chosen as an example of a test case in which there was a variation between the worst and best solutions found in previous experiments (see table 4). Although the best solution found remains the same in every experiment, the average fitness peaks

Table 7

Effect of population size on minimum, average and best fitness found using GA_{S1} for Tuesday 22/4.

Population size	Minimum fitness	Average fitness	Best fitness
1	0.053	0.170	0.333
20	0.167	0.296	0.333
40	0.167	0.296	0.333
60	0.25	0.313	0.333
100	0.25	0.321	0.333
200	0.25	0.313	0.333
500	0.167	0.281	0.333

at a population size of 100. Interestingly, even with a population size of 1 we are able to find the best solution in 35 of the trials. We note that with a population size of 1, the genetic crossover operator is obviously of no value. Any improvement to the initially randomly generated solution must have been found by mutation alone. We are only performing local search, i.e. *hill-climbing*. This experiment was not repeated with other test cases.

A series of experiments that varied the mutation rate was also performed and showed that the best result obtained was not affected at all by choice of mutation rate, although the rate did have a small effect on the average result. We conclude that this particular GA is remarkably easy to tune in order to obtain satisfactory results.

8. GA_{Timer} – Scheduling the factory arrivals

The best squad assignment found by GA_{S1} is fed to a timing GA, GA_{Timer}, which actually creates the schedules by searching the space of all start times for each squad.

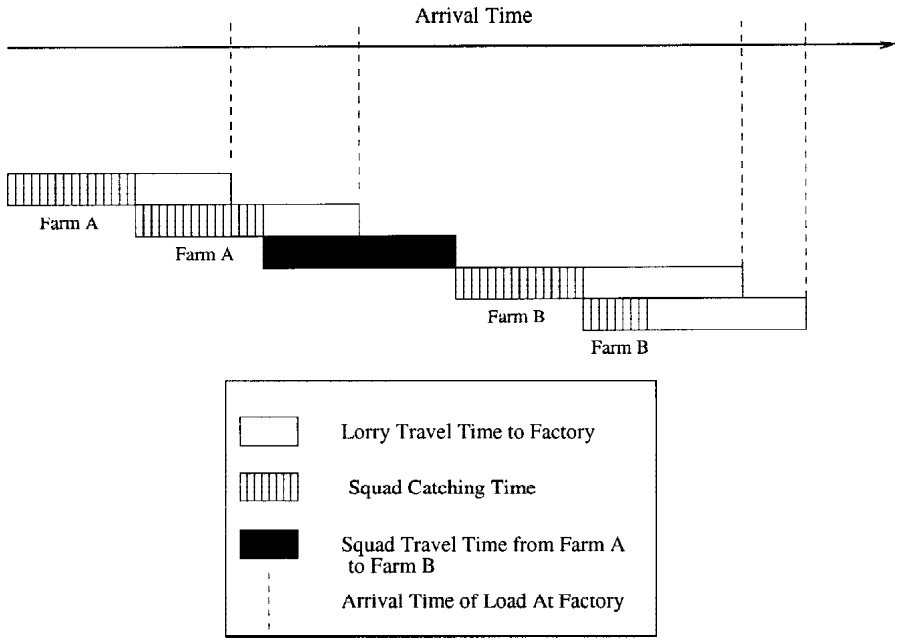

Figure 3. An example schedule for a squad.

As discussed in section 4, allocating a start time to a squad will automatically dictate the arrival time of each of its loads, as it cannot take any break while working. A sample schedule is illustrated in figure 3 for a squad that must catch 2 lorry loads at farm A, followed by 1.5 lorry loads at farm B. The diagram shows the time spent catching each of the loads, each of which is immediately collected by a lorry and delivered to a factory. Once the squad has finished catching its loads at farm A, it immediately travels to farm B where it catches a further 1.5 loads. Note that the half load takes a shorter time to catch. If a squad visits any farm that is classified as "diseased", the farms it visits must first be arranged in a sequence in which the squad does not travel from a diseased farm to a disease-free one before applying GA_{Timer}. This is the only "hard" sequencing constraint, though the squads may have preferences as to the order in which they visit farms – for instance, they prefer to do the last catch of their shift at a farm located near to their base factory. At present, only the hard sequencing constraint is considered.

The genome has N_s genes, one for every squad used in the schedule. All arrivals at the factory are timed to the nearest 15 minutes. Each gene can take a value in the range 0–88. This is interpreted such that gene i having value a means that squad i has a start time, in minutes after midnight, of (15 a). The upper limit of 88 equates to a start time of 22:00 hours. This is due to the fact that the last time a factory can receive a load is 22.30 hours – as the minimum time for catching a load is 30 minutes, a squad cannot start at a time later than 22:00 hours. (Some farms are located immediately outside the factory and therefore there is no travelling time between farm and factory.)

8.1. Reducing the search space

We can pare down the search space somewhat by incorporating some knowledge of the type of schedule we hope to achieve. For instance, GA_{S1} has already assigned squads to catch the birds that satisfy the Coupar Angus requirement for their opening and closing loads (see section 6.2). Therefore, we can narrow the range of starting times for these squads so that their relevant loads arrive within the required time spans, taking into account the distance between each farm and factory.

We can also reduce the time range of several other squads. Part-time squads do not start work before 9 a.m. in the morning – on the other hand, a full-time squad must have finished work by late afternoon. Hence, a unique range (a, b) can be defined for each squad, considerably smaller than the (0, 88) range initially proposed.

The population is initialised by selecting a random value from the specific range assigned to a gene. The mutation must be modified to ensure that a gene value is mutated to something in the correct range for the gene. Similar approaches to "inoculation" or non-random initialisation of the population have been taken by others, e.g. [21], and have proved effective.

8.2. Defining the quality of a schedule

Using the squad start time indicated on the genome, a complete arrival schedule can be built for each factory. A lorry schedule is built by initially assigning a new lorry to every load that must be collected, assuming an infinite resource of lorries. When the complete lorry schedule has been created, a reduction in lorry numbers can be performed by examining all lorry start and finish times – if the finishing time of task i is earlier than the start time of task j, then they can both be delivered by the same lorry. This procedure is automated and results in a substantial reduction in lorry numbers once performed.

The quality of the resulting arrival and lorry schedules is again determined via a fitness function that checks the schedule for violations of a set of constraints. A schedule is penalised if:

- Loads arrive after the factory closes.
- Too many loads arrive before the factory opens.
- More than 1 load arrives at the same time.
- More lorries are needed than are available.

In addition, the standard deviation of the gap between arrival times of loads is measured and this is added to the penalty to try to ensure an even spread of load arrivals throughout the day. Each constraint is weighted and multiplied by the number of violations to give the final penalty. Once again, satisfactory penalty values proved straightforward to find.

8.3. Results

GA_{Timer} was able to evolve schedules that satisfied all of the above constraints and also spread the distribution of loads evenly throughout the day in approximately 2000 evaluations. We found that the number of vehicles required to execute the schedules was *fewer* than the number presently used by the factory. This could represent a substantial cost saving to the company, but also means that while operating with its current vehicle fleet, it has a contingency of vehicles for coping with any unforeseen circumstances such as a lorry breaking down or being off the road for servicing. Currently, the company must rely on outside contractors to provide cover in such circumstances.

9. Conclusion

We have presented the results from tackling the real-world scheduling problem of a chicken processing company using a genetic algorithm. By decomposing the problem into separate sub-problems and tackling each with a separate GA, we have shown that we can provide a fast and reliable scheduling system that can be easily used by the company. The system produces schedules very similar to those currently produced by hand at the company and hence meets their requirement that the schedules produced are compatible with current working practices at the factory.

We find that a strategy-based approach resolves many of the problems encountered in previous work (not reported here) that attempted to use a GA to directly evolve a solution to chicken scheduling. Although GA_{S1} requires a high processing effort in that each section of the chromosome requires different crossover and mutation operators, it reliably produces high quality schedules on all cases tested. The general robustness of GA_{S1} indicates that by introducing flexibility into the GA by allowing a choice of heuristics, we have transformed the search space into something that is easily traversed and well correlated. Also, by exploiting some domain knowledge in the chromosome, we not only guarantee the minimum quality of a solution, but also sufficiently reduce the size of the search space to ensure an efficient search takes place.

As a final comment, we suggest that although a GA can often be outperformed by other non-evolutionary algorithms over a range of scheduling problems, for example [8], there may exist a niche for GAs in tackling complex, real-world problems. For example, Rana et al. [16] point to a niche for GAs in fast, stochastic and large, real-world scheduling problems, and Fang [5] points to a general niche for GAs in larger scheduling problems with quality criteria which are notoriously awkward for heuristic approaches. The strategy-oriented approach described may turn out to be a useful method for tackling complex problems where it is hard to define exact heuristics for building schedules. Clearly, such a representation could also be used in simulated annealing or hill-climbing, and the authors are currently investigating whether such techniques have any advantages or disadvantages over the genetic algorithm in this

particular case. We also intend to investigate whether more conventional constraint satisfaction algorithms can achieve the same results.

We note, however, that the use of a non-deterministic, population-based scheduling algorithm may be welcomed by a production manager, as he or she can be provided with several different schedules, all satisfying the same constraints, and hence can make a personal choice as to which best suits the current circumstances.

Acknowledgements

The first and third authors acknowledge the support of EPSRC Grant No. GR/L22232.

References

[1] C. Bierwirth, A generalized permutation approach to job-shop scheduling with genetic algorithms, OR Spektrum 17(2–3)(1995)87–92.

[2] R. Bruns, Direct chromosome representation and advanced genetic algorithms for production scheduling, in: *Proceedings of the 5th International Conference on Genetic Algorithms*, ed. S. Forrest, Morgan Kaufmann, San Mateo, February 1993, p. 352.

[3] L. Davis, Job shop scheduling with genetic algorithms, in: *Proceedings of the International Conference on Genetic Algorithms and their Applications*, ed. J.J. Grefenstette, Morgan Kaufmann, San Mateo, 1985, pp. 136–140.

[4] L. Davis, Adapting operator probabilites in genetic algorithms, in: *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, ed. J.D. Schaffer, Morgan Kaufmann, San Mateo, 1989, pp. 61–69.

[5] H-L. Fang, D.W. Corne and P.M. Ross, A genetic algorithm for job-shop problems with various schedule quality criteria, in: *Evolutionary Computing: 1996 AISB Workshop: Selected Papers*, ed. T.C. Fogarty, Lecture Notes in Computer Science 1143, Springer, 1996, pp. 39–49.

[6] H-L. Fang, P. Ross and D. Corne, A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems, in: *Proceedings of the 5th International Conference on Genetic Algorithms*, ed. S. Forrest, Morgan Kaufmann, San Mateo, 1993, pp. 375–382.

[7] E.D Goodman, A.Y Tetelbaum and V.M Kureichik, A genetic algorithm approach to compaction, bin packing and nesting problems, Technical Report, Case Center for Computer-Aided Engineering, Michigan State University, 1994.

[8] A. Juels and M. Wattenberg, Stochastic hillclimbing as a baseline method for evaluating genetic algorithms, Technical Report, UC Berkeley, 1994.

[9] B. Kroger, Guillotineable binpacking: A genetic approach, European Journal of Operation Research 84(1995)645–661.

[10] W.B. Langdon, Scheduling planned maintenance of the national grid, in: *AISB Workshop on Evolutionary Computing 1995*, ed. T.C. Fogarty, Springer, Berlin, Germany, 1995.

[11] B. McCarthy, S. Crawford, C. Vernon and J. Wilson, How do humans plan and schedule, presented at *The 3rd Workshop on Models and Algorithms for Planning and Scheduling Problems*, 1997.

[12] G.F Mott, Optimising flowshop scheduling through adaptive genetic algorithms, Master's Thesis, Chemistry Part II Thesis, Oxford University, 1990.

[13] J.F. Muth and G.L. Thompson (eds.), *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[14] R. Nakano and T. Yamada, Conventional genetic algorithms for job shop problems, in: *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, 1991, pp. 474–479.

[15] G. Niemeyer and P. Shroma, Production scheduling with genetic algorithms and simulation, in: *Parallel Problem Solving from Nature: PPSN IV*, eds. Manner, Davidor and Schwefel, Springer, 1996, pp. 931–939.
[16] S. Rana, A. Howe, K. Mathias and D. Whitley, Comparing heuristic, evolutionary and local search approaches to scheduling, in: *The 3rd Artificial Intelligence Planning Systems Conference – AIPS-96*, 1996.
[17] J.T Richardson, M.R. Palmer, G. Leipin and M. Hilliard, Some guidelines for gas with penalty functions, in: *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, ed. J.D. Schaffer, Morgan Kaufmann, San Mateo, 1989, pp. 191–197.
[18] J.D. Schaffer, R.A. Caruana, L.J. Eshelman and R. Das, A study of control parameters affecting online performance of genetic algorithms for function optimisation, in: *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, ed. J.D. Schaffer, Morgan Kaufmann, San Mateo, 1989, pp. 51–61.
[19] J. Shaw, References on the application of genetic algorithms to production scheduling, available via anonymous ftp site cs.ucl.ac.uk, file genetic/biblio/ga-js-sched-bibliography.txt, June 1994.
[20] A.E Smith and D.M Tate, Genetic optimization using a penalty function, in: *Proceedings of the 5th International Conference on Genetic Algorithms*, ed. S. Forrest, Morgan Kaufmann, San Mateo, 1993, pp. 499–505.
[21] P.D. Surry and N.J. Radcliffe, Inoculation to initialise evolutionary search, in: *3rd AISB Workshop on Evolutionary Computing*, Springer, 1996.
[22] D. Whitley, A genetic algorithm tutorial, Technical Report, Colorado State University, March 1993.