A Simulation Based Genetic Algorithm Workforce Scheduler

Group Activity Reflections

Our task through the group activities was to identify potential areas where the Port of Liverpool could be improved using simulation and optimisation. In the first lab we had to collect ideas for project, based on reports and research papers that have been conducted in the field. I have suggested the idea of simulating the effect of noise barriers around the harbour to improve the lives of people living close to the port. In the focus group, in the following lab I took the role of the project lead and technical lead. Sadly, the idea that I suggested did not make the cut following the focus group. In the last lab we had to present our chosen projects to the class. My task in the presentation was to introduce our "consultancy", so I have chosen some real-world projects that could have used simulation to plan the project.

Overall, I think our group did well in these exercises, and I believe that I have contributed to this success. However, looking back at these events, I think I could have taken a bigger role in the group in general, and I could have contributed more in the discussions in the focus group.

Conceptual Model

Our task was to provide a fast-food chain a simulation-optimisation tool that can make workforce management more efficient. To solve this problem the following conceptual model was developed.

Objective: The purpose of the model is to simulate the efficiency of a weekly schedule, providing rich metrics that can be used to measure the quality of the schedule. The entire simulation should be fast enough to be able to run multiple times a second to allow for optimisation.

Inputs: The most important input to the simulation is the weekly schedule. This schedule can be used to find which employee is assigned to what role in which restaurant for any opening hour.

The simulation also requires a company database that contains 1) a list of employees including their wage, and a measure of their skill in each role, 2) a list of restaurants including their estimated popularity for the planned week.

Outputs: The most important output of the simulation is the estimated profit and an approximation for customer satisfaction. The simulation should use diagrams to display the change of these values over time, as well as for contributing values such as cleanliness, queue size, etc.

Content: The model contains three core agents: Employees, Restaurants, and Customers.

Each Employee receives their own schedule derived from the input to the model. Employees can then decide when they go to serve a shift at the correct Restaurant. Their behaviour can be described via the state chart in Figure 1.

Figure 1 Employee state chart

A simplification is applied here, whereby unplanned leaves are not accounted for. When employees work, their efficiency is described by their skill and their "attentiveness" which decreases the more they have worked during a day. For example, a more skilled Employee will provide better service at the counter as opposed to a less skilled one. However, someone just starting a shift with less skill can do a better job than someone with a higher skill, but who's been working for 8 hours.

Customers' behaviour in a Restaurant is modelled by the following queuing system, in Figure 2.

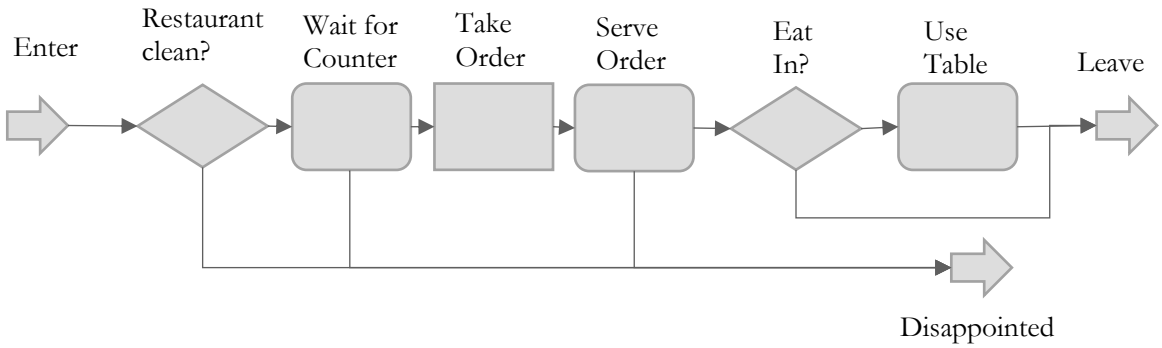

Figure 2 Restaurant queuing system

Let us examine this system, with the eyes of a Customer. First, if the Restaurant is unclean they just leave disappointed. Next, if the Restaurant is clean enough they stand in a single queue for all counters. When it is their turn an Employee takes their order at a counter. Passing the order to the kitchen it is prepared within a few minutes. If they have to queue for more than 15 minutes they leave. After picking up the order, they can decide to eat in or take out. If they decide to eat in then a table will be used for the time the order is consumed. Either way, before leaving the cleanliness of the Restaurant is decreased.

Finally, Customers are simple agents that take a passive role in the model. They are responsible for grading their satisfaction with the service provided. The satisfaction is measured by a weighted sum of the food and service quality, the time taken to be served and the cleanliness of the Restaurant. This formula was based on findings in [1], [2].

Assumptions: Several assumptions are made to make this model possible. The model assumes that Employees have a predictable efficiency that is not affected by outside factors. It is also required that any Employee can attend shifts in any Restaurant.

Restaurants are assumed to all work in the same manner. There is one manager, several kitchen, counter, and cleaner staff. The manager ensures that all operations run smoothly. The kitchen staff prepares orders in front of the Customers. Counter Employees take orders and take payments. Cleaners ensure that the tidiness of the Restaurant is maintained. It is assumed that Restaurants have 5 counters with a single queue for them, of a capacity of 50. All Restaurants are assumed to have 30 seats.

Customers are assumed to arrive at a predictable rate, that can be derived from past data. Customers have a predictable order size that adheres to some distribution.

Simplifications: To simplify Employees it is assumed that they will not be late or be absent from work.

The Restaurant model can be simplified by having all of them with the same size, configuration and opening times. The only difference between them is their relative popularity.

Customers are simplified to queue in an orderly fashion, and to leave promptly when they finish eating. It is also assumed that they will only leave if they have to wait more than some threshold time. Furthermore, Customers are simplified to be only a population where individuals do not make decisions or have a memory of previous visits.

This model was based on the operations of a fast-food restaurant described in [3].

IMPLEMENTED MODEL

To implement the Restaurants' queuing system (see Figure 3) AnyLogic's Process Modelling Library was used. The core building blocks to this system are the Seize, Release, Service and Delay nodes.

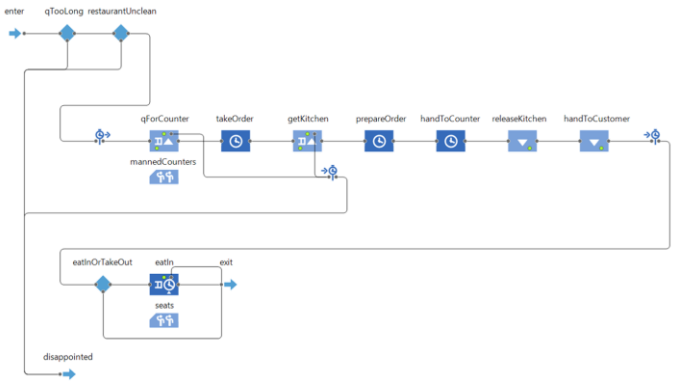

Figure 3 AnyLogic Restaurant process flow

This implements the conceptual model; the extra complexity is only due to the software's limitations. To avoid exceeding the queue capacity of the *qForCounter* node an additional branch was added to check this beforehand. The service time is measured on the critical path for Customers, between a Time Measurement Start and End node. Disappointed customers divert from this critical path therefore an additional Time Measure End node is needed.

As soon as the Customer enters the counter queue they decide the order size based on a distribution. When a Customer Seizes a counter or kitchen Employee, then this assignment is stored in the Customer agent, as well as sending the appropriate messages to the Employee agent, which ensures that they are in the right state. The Employees' skill levels define the wait times in the subsequent Delay nodes, and the quality of the service. The "payment" only happens when the food is received, i.e. at the Hand to Customer node. When leaving at the Exit node, the Customer rates the experience by updating its satisfaction score.

The Customers enter the restaurant based on the predicted popularity in the company database. Since data was not readily available for this (Category B), it was estimated from Google Maps' popularity feature [4] of a business similar in location, size, and operation. To improve the simulation's applicability, this data should be collected from the Restaurants and used instead of the estimates. This information was normalised to a [0, 1] range where 1 is the most popular time frame and all others are relative to this. Individual Restaurants' popularity rates are derived from this matrix, multiplied by a constant factor, i.e. some restaurants are more popular than others, but they have the same change curves.

Employees' core logic was encapsulated as a state chart in Figure 4. Some internal states were added over the conceptual model, to allow for more granular control of the agents.

The transitions are driven by events. A built-in Schedule object is generated for each Employee, derived from the input of the simulation. This Schedule can be queried to see if there are any assignments for a given hour. When an assignment starts the agent decides which internal state to occupy within the working state. The internal states are then updated via messages from the Restaurant's process flow.

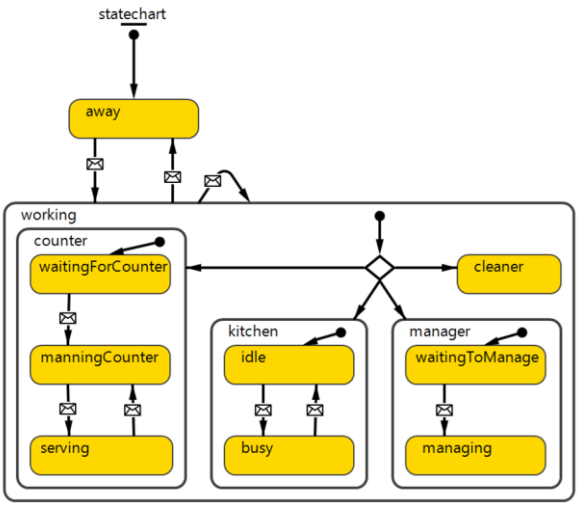

Figure 4 AnyLogic Employee state chart

As a simplification, Employees "claim" their wage immediately after a shift. This also makes it possible to track the profit of restaurants in real time, which is an interesting performance indicator to visualise. To give the optimisation process an incentive for using employees that live close to the restaurant, a "travel reimbursement" is given to employees as part of their wage. The amount is derived from the distance to the Restaurant.

Three limitations were assumed that require special handling within the two agents:

1. Restaurants have only 5 counters, therefore if any more Employees are assigned to the role, they have to wait to take their positions.
2. There can be only one "acting" manager in a Restaurant, therefore if any more Employees are assigned to the role, they have to wait to take their positions.
3. Changing roles immediately is possible but not changing Restaurants.

For the first two limitations, auxiliary processes were created (in the Restaurant) where Employees queue to assume their assigned positions. The third limitation is an assumed business rule, requiring the schedule-maker to follow this.

Customers are simple agents, that do not make decisions. Customers are represented as a pool of agents in the world. When a customer is required to enter a restaurant (as dictated by the popularity of the Restaurant), one is drawn from the pool; when they leave the Restaurant, they are inserted back in the pool.

One of the key assumptions in the Customer model is the distribution of order sizes. As this information is not readily available (Category B), the distribution in Figure 5 was used instead, where 1 corresponds to a normal size order 2 corresponds to double size, etc. As soon as hard data is available on the real distribution of this, it should be used instead of this estimate to improve the model's accuracy.

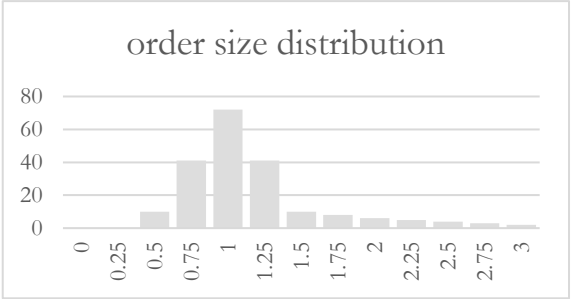

Figure 5 Order size distribution

Another important role of customers is to evaluate the service in terms of their satisfaction. To measure this, the following formula was used:

$$satisfaction \mathrel{+}= \begin{cases} -3 & if\ disappointed \\ 1.0\,f + 0.9\,c + 0.7\,s + 0.6\,T(t) & otherwise \end{cases}$$

where $f = food\ quality, c = cleanliness, s = service\ quality, t = time\ to\ served$

and $T = time\ satisfaction$ shown in Figure 6. Again, this is only an estimate that should be replaced with data collected from real customers.

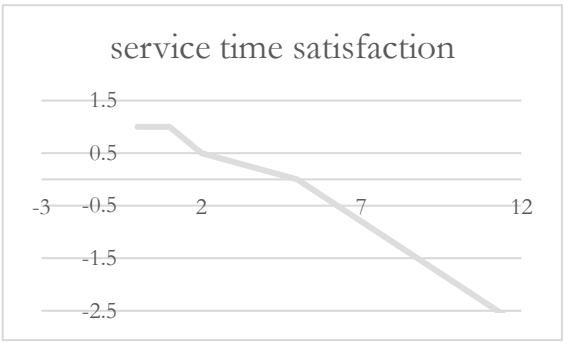

Figure 6 Time satisfaction

Optimisation Experiment

Since the built-in optimiser of AnyLogic is not capable of combinatorial optimisation, that is required to solve a workforce scheduling problem, a custom genetic algorithm was implemented. For a quick introduction to genetic algorithms please refer to [5].

The algorithm was customised in several ways to fit the problem domain. The gene encoding is similar to the one used in [6]. The gene consist of a list of assignments, for each employee, for each day. A single assignment can be represented as a tuple of (r, s, p, q) where $r = restaurant\ worked\ [1, 5],\ s = role\ [1, 4],\ p = start\ time,\ q = end\ time$. It is crucial that the algorithm maintains the validity of the schedule, that is, no overlaps are allowed, and employees cannot change Restaurants between two shifts.

The optimisation was designed to be able to use any measure from the simulation as a fitness function. The fitness function that performed best during the development of the model was a simple function that can be defined as $f = profit + \sum_{c \in customers} satisfaction_c$.

There are no hard or soft constraints for the algorithm, therefore it is free to explore the search space. Instead, the mutation operators were created so that they help to maintain some properties that are akin to soft constraints. An improved version of the system could use penalties to account for constraints such as planned absence, or preferred days.

The crossover operator is a standard uniform crossover operator over daily schedules. For each employee and each day, there is a 50% chance of taking that day's schedule from the father or mother chromosome. The inverse selection is not discarded, instead it is used as additional offspring. This operator can be visualised by the following diagram in Figure 7:

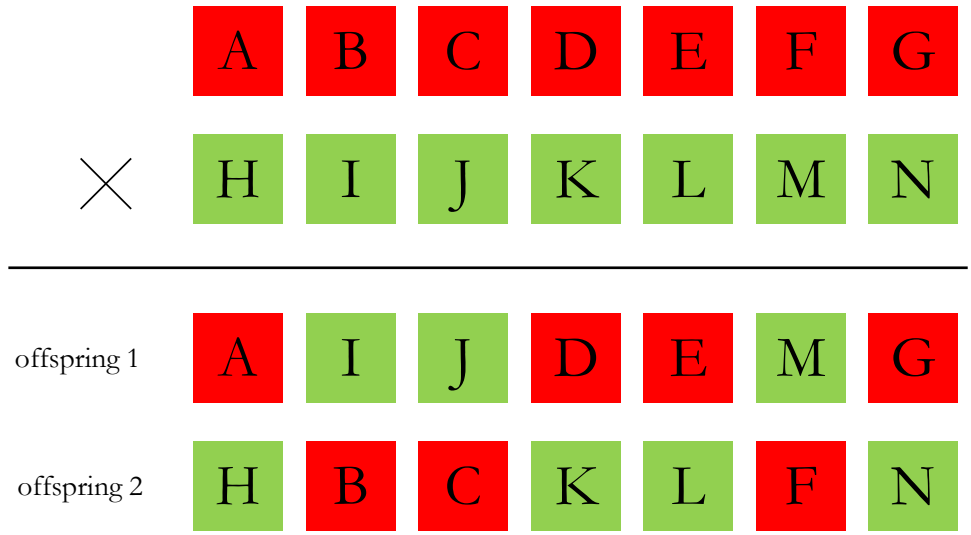

Figure 7 Uniform crossover operator. Each column corresponds to a day. E.g. in offspring 1 the first day comes from the first parent the second and third comes from the second parent, etc.

This operator maintains the invariants of the schedule (no overlap, no restaurant changes). Furthermore, it maintains employee parity, i.e. individual employees' schedules are derived from different versions of the same employee's schedules. This way if a certain solution can use an Employee's time efficiently this can carry over to the next generation.

The selection method was chosen to be a rank-based roulette wheel selection, as it can scale the results of the fitness function automatically, and it reduces premature convergence most efficiently as evidenced by [7]. Elitism in selection is a tuneable factor in the optimiser settings.

There are 11 mutation operators organised into three sets:

A. Exploratory
These operators navigate the search space in an unguided fashion, increasing diversification. There is 20% percent chance to use one of these operators for every mutation.

1. *Swap days*: Selects two days within an employee's schedule and swaps them.
2. *Change shift length*: Either extends or contracts a shift by some hours. Complex logic is used to maintain the invariants of the schedule. E.g. it is ensured that a schedule is only extended if it is not bounded on both sides (no overlap). If a shift is selected that cannot be mutated then the schedule remains unchanged after this mutation.
3. *Change role*: Randomly picks an assignment and changes its role, or s value. The role selection is biased toward ones the employee is more skilled in.
4. *Change restaurant*: Pick an assignment and change the restaurant assignment, or r value. To ensure that no immediate restaurant change happens, it is ensured that the assignment is not bounded by other assignments. The restaurant selection is biased toward ones the employee lives closer to.
5. *Split shift*: Splits an assignment into two assignments with different roles. The new role is biased using the employee's skills.
6. *Swap schedules*: Select two employees and swap their entire schedules. Employees with similar skills are more likely to be selected.

B. Constructive
These operators try to fix common inconsistencies found in random individuals. There is 70% chance to choose one of these operators for every mutation.

7. *Add shift to empty day*: Select an employee at random. If they have a free day then assign a random shift to it.
8. *Bridge idle time*: Finds gaps in employees' schedules where someone works for a few hours, then finishes that shift, but have to come back to the same restaurant later that day. Since this operation has many preconditions, it is common for it to leave the chromosome unchanged. E.g. if an employee is selected that only takes one assignment a day then nothing can be done.
9. *Fill restaurant requirement*: The employee-oriented schedule is converted into a restaurant-oriented one. From this intermediate schedule the number of assigned workers for any hour can be queried. For a random day it is checked whether the assignment requirements are met: at least 1 manager and cleaner, and at least 3 counter and kitchen staff. If at any hour this is not met, then an employee is selected that is not working that day and they are assigned a position that fulfils this requirement. This is just an arbitrary requirement, if there are any business rules that suggest different requirements, then that should be substituted here.

C. Destructive
These operators ensure that the schedule is never fully saturated, giving space for other operators to explore the search space. There is 10% chance to choose one of these operators for every mutation.

10. *Delete shift*: Pick a random assignment and remove it from the employee's schedule. Longer shifts are more likely to be removed.
11. *Clear schedule*: Pick a random employee and clear their entire weekly schedule.

An additional operator is used that differs from a standard genetic algorithm: mass extinctions (as suggested by [8]). It is possible at each generation to replace some percentage of offspring with random individuals. The chance for this can be tuned or turned off in the settings. This operator ensures diversity in the population throughout the search.

To integrate this algorithm into AnyLogic the Parameter Variation Experiment was extended. At the start of each iteration a schedule (chromosome) is requested from the GA. The genetic algorithm keeps track of which ones need evaluation. If all individuals have been evaluated in a generation then a new generation is created and the first individual is returned. This method only works if individuals are evaluated sequentially, therefore parallel execution was disabled in the Experiment. On the other hand, the UI of the experiment makes it possible to change some of the search parameters on-the-fly, which makes it possible to tune it as its running.

Sadly, in the end this search algorithm was not capable of producing high quality schedules. With some further work, it might be possible to achieve this level, though. An improvement could be to simplify the model, to make it evaluate faster, as now it takes many seconds to evaluate a generation. Another change could be to analyse the efficiency of mutation operators and fine-tune their selection probability. Lastly, other fitness functions should be evaluated that can guide the algorithm better.

REFERENCES

[1] A. K. Law, Y. V. Hui, and X. Zhao, 'Modeling repurchase frequency and customer satisfaction for fast food outlets', *International journal of quality & reliability management*, vol. 21, no. 5, pp. 545–563, 2004.

[2] C. Park, 'Efficient or enjoyable? Consumer values of eating-out and fast food restaurant consumption in Korea', *International Journal of Hospitality Management*, vol. 23, no. 1, pp. 87–94, 2004.

[3] R. R. Love and J. M. Hoey, 'Management science improves fast-food operations', *Interfaces*, vol. 20, no. 2, pp. 21–29, 1990.

[4] Google, 'Reference restaurant'. [Online]. Available: https://goo.gl/maps/TcJKbX8ewvB2. [Accessed: 30-Apr-2018].

[5] M. Srinivas and L. M. Patnaik, 'Genetic algorithms: A survey', *computer*, vol. 27, no. 6, pp. 17–26, 1994.

[6] J. Tanomaru, 'Staff scheduling by a genetic algorithm with heuristic operators', in *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, 1995, vol. 3, pp. 1951–1956.

[7] N. M. Razali and J. Geraghty, 'Genetic algorithm performance with different selection strategies in solving TSP', in *Proceedings of the world congress on engineering*, 2011, vol. 2, pp. 1134–1139.

[8] T. Krink and R. Thomsen, 'Self-organized criticality and mass extinction in evolutionary algorithms', in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, 2001, vol. 2, pp. 1155–1161.