CHALMERS
UNIVERSITY OF TECHNOLOGY

Simulation of Complex Systems

Supply chain optimization using a genetic algorithm and agent-based models

Credi, Jacopo
jacopo@student.chalmers.se

Henoch, Martin
henochm@student.chalmers.se

Meinel, Jonas
meinel@student.chalmers.se

Wänerlöv, Viktor
vanerlov@student.chalmers.se

December 2015

Abstract

The purpose of this project was to design and optimise the flow of a supply chain network with different methods (agent-based models) which use local information about the system and investigate how they compared to a method using global information (a genetic algorithm). The latter was implemented by extending an existing method found in the literature, and used as a benchmark for the agent-based model because its outcome should be close to the optimal flow.

The underlying supply chain consisted of different facilities with different geographical locations on a continuous square space, and a measure of the supply chain profit was defined and used for comparing the models. Two different agent-based models were designed: a model based on simple reinforcement learning, with a trade-off between greed and fidelity between facilities, and a more complicated model inspired by the market. The model dependence on (some of the) parameters was also investigated, but particular emphasis was placed on engineering the models and their dynamics such that their long-term steady-state corresponds to a highly profitable configuration.

Contents

1 Introduction

A supply chain can be viewed as a network with goods flowing from suppliers to customers through several intermediate stages consisting of many facilities including e.g. retailers, warehouses and manufacturers. Traditionally, marketing, distribution, planning, manufacturing, and purchasing organizations along the supply chain operated independently. In the last few decades, however, operation researchers have focused their attention on the design and performance analysis of the supply chain as a whole, developing the concept of Supply Chain Management (SCM). In particular, the problem of designing an optimal Supply Chain Network (SCN) is one of the most comprehensive strategic decision problems that need to be addressed for long-term efficient operation of the whole supply chain [3, 5].

Project goals

The problem of SCN optimisation includes determining optimal number, location, capacity and type of plants, warehouses and distribution centres to be built, as well the amount of raw materials to consume, products to produce and items to ship among the network nodes. In this project, however, we have focused on one of these sub-problems: a fixed number of facilities are placed in a a 1×1 square space, with fixed demand and supply capacities, and we consider the problem of optimally forming a supply chain network, i.e. of shipping items across the network so that the profit of the supply chain is maximised.

- Our first goal was to design and implement a genetic algorithm for SCN optimisation, inspired by some previous works in the field [1, 4], and in particular extending the approach of Chu and Beasley [2].
- Secondly, we wanted to create, implement and simulate some agent-based models, in which facilities only have a partial knowledge of the environment and interact with each other in discrete time-steps. We experimented with different kinds of interactions and explored parameter spaces with the goal of observing some sort of self-organization of the facilities into a steady-state corresponding to a highly profitable supply chain network.

The common mathematical framework

A set of facilities is placed on the unit square. Each facility can belong to one of K different types/layers, and there are N_k facilities in layer k, with $k = 1, \ldots, K$. Each facility of type 1 ("end-customers") is assigned a fixed scalar value x_j, with $j = 1, \ldots, N_1$, representing facility j's demand of products. Similarly, facilities of type K ("end-suppliers") are characterised by a fixed scalar value s_j, with $j = 1, \ldots, N_K$, representing supplier j's output capacity.

Each facility of type $k = 1, \ldots, K-1$ can be connected to (at most) one facility of type $k+1$, representing a shipment between the two facilities. Once connections are determined, we have a weighted undirected K-partite graph $G = (F_1, F_2, \ldots, F_K, E)$ which corresponds to the supply-chain network of facility sub-sets $F_1, F_2, \ldots, F_K$, connected by edges set E. Edges e_{ij} are weighted, with the weight being some function $f(d_{ij})$ of the the Euclidean distance d_{ij} between the connected nodes i and j in the unit square.

The problem of optimizing the supply chain can therefore be solved by finding the set of edges E^* that optimizes some objective function measuring the network performance. This objective function can of course be defined in many different ways and throughout the project we have tried a few of them. In particular, we found that a good objective function should somehow reflect the supply chain profit. To a first approximation, we can measure the profit as total amount of items sold to end-customers (i.e. the flow

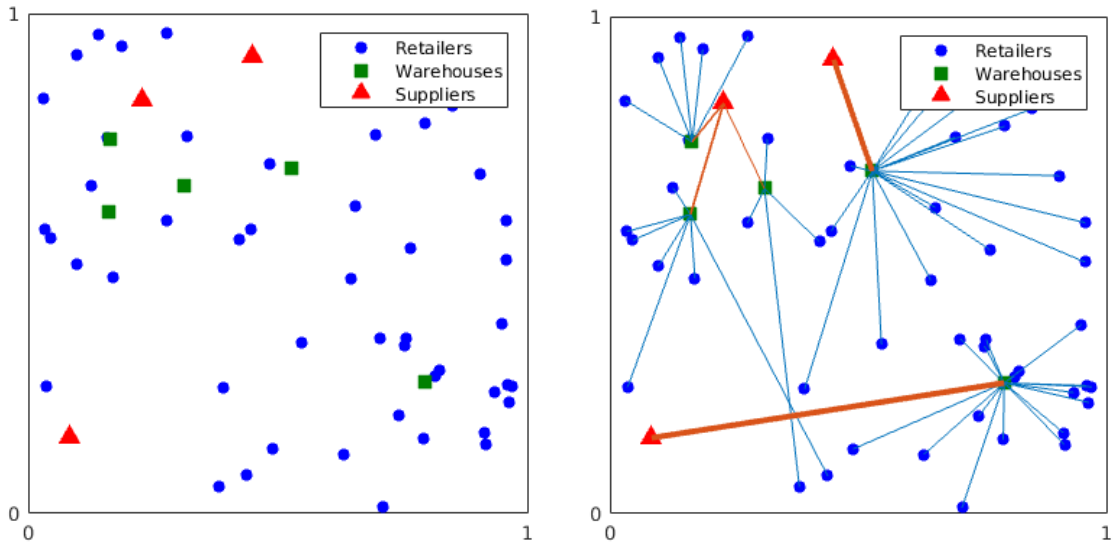

Figure 1: *An example of a random map. Three kinds of facilities ($K = 3$) are placed on the unit square, color-coded, before (left panel) and after (right panel) adding edges.*

of products in the network) minus the shipment costs:

$$f(E) = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \theta_{ij}^{[1]} \; x_i^{[1]} \; - \alpha \; \sum_{k=1}^{K-1} \left(\sum_{i=1}^{N_k} \sum_{j=1}^{N_{k+1}} \theta_{ij}^{[k]} \; x_i^{[k]} \; w_{ij}^{[k]} \right) , \tag{1}$$

where:

- $x_i^{[k]}$ is the demand of customer i in layer k;
- $\theta_{ij}^{[k]} = \begin{cases} 1 & \text{if edge } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$, and refers to edges between layers k and $k+1$;
- $w_{ij}^{[k]}$ is the weight of the link (if any) from node i in layer k to node j in layer $k+1$;
- α is a parameter which can be interpreted as the transportation cost of one unit of product per unit distance.

Note that demands $x_i^{[k]}$ are only specified at the beginning (and kept fixed) for the first layer $k = 1$. Demands of facilities belonging to inner layers $k = 2, \ldots, K-1$ are in fact determined by the amount of orders received from the facilities in layer $k-1$ ("below"):

$$x_i^{[k]} = \sum_{j=1}^{N_{k-1}} \; x_j^{[k-1]} \; \theta_{ji} \; . \tag{2}$$

Facilities belonging to the "top" layer K supply facilities in layer $K-1$ according to their supply capacity s_i and their customers' demands $x_i^{[K-1]}$. Therefore, the optimisation of $f(E)$ is subject to the following set of constraints:

$$\sum_{j=1}^{N_{K-1}} \; x_j^{[K-1]} \; \theta_{ji} \; \leq s_i \; , \quad \text{for} \quad i = 1, \ldots, N_K \; . \tag{3}$$

2 A genetic algorithm for SCN optimisation

Simple case: 2-layers supply chain

In the simplest case $K = 2$ the problem of generating an optimal set of edges is equivalent to the *generalised assignment problem*, a well defined problem in combinatorial optimization which can be stated as follows:

Instance A pair (T, W), where T is a set of tasks and W is a set of workers. Each worker $i \in W$ has a capacity w_i and each worker-task pair (i, j) is associated a size s_{ij} and a profit p_{ij}.

Objective Assign a subset T_i of the tasks to each worker i such that the total profit is maximised. The assignment should be feasible, i.e. $\sum_j s_{ij} \leq w_i$, where $j \in T_i$.

In our case, clearly, customers correspond to tasks, and suppliers to workers performing the tasks. Costs of tasks/deliveries correspond to customers' demands and capacities of the workers/suppliers to their supply capacity. The profit obtained by an agent/supplier performing a task/delivery to a customer depends on the distance between the supplier and the customer.

Chu and Beasley [2] proposed a heuristic approach to solve the generalised assignment problem, based on a GA. Chromosomes are encoded as vectors of length N_1 (number of tasks), with the j^{th} entry being the index i of the worker that task j is assigned to (or, in our case, the index i of the supplier shipping to customer j). An entry can of course be empty, meaning that that task is not performed by any worker (or, in our case, that customer receives no supply). See Figure 2 for a graphical representation of this encoding scheme.

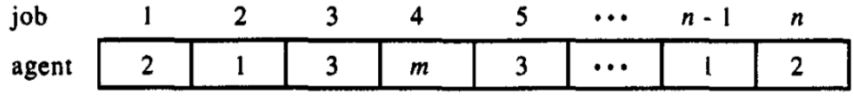

Figure 2: *Graphical representation of an individual's chromosome, from [2]. In this example, task 1 is assigned to worker 2, task 2 to worker 1, task 3 to worker 3, and so on.*

One key aspect of this heuristic approach is the way it deals with the capacity constraint, by evaluating an "unfitness" measure of each solution, besides its fitness. The unfitness of solution l is defined as

$$u_l(E) = \sum_{i=1}^{N_2} \max\left[0,\ \sum_{j=1}^{N_1} x_j^{[1]} \theta_{ji} - s_i\right], \tag{4}$$

so that $u_l = 0$ if and only if solution l is feasible. Infeasible solutions ($u_l > 0$) are kept in the population, in order to allow a better exploration of the configuration space, but the solution with maximum unfitness is discarded at each steady-state replacement step. Although this procedure is not guaranteed to produce feasible solutions, it actually does it quite nicely and quickly in practice.

We have implemented a simplified version of the method described in [2] (without problem-specific operators) to find a close-to-optimal network configuration in the simple case $K = 2$, using the fitness function defined in Eq. (1).

3-layers supply chain

Although the 2-layer case is simple and equivalent to a well-defined problem in combinatorial optimization, we need at least 3 layers in order to have a nontrivial supply chain model. With this goal in mind we generalised the GA approach to solve a "multi-assignment"[1] problem, with several levels of assignment: tasks are assigned to workers at the lowest assignment level, then workers are assigned to upper-level structures,

[1]This name is to distinguish this problem from the multi-level generalized assignment problem, a different combinatorial optimization problem in which agents can work at multiple levels of efficiency.

and so on. In our case, with $K = 3$, "retailers" (level-1 customers) are assigned to "warehouses" (level-1 suppliers), which in turn (as level-2 customers) are assigned to "manufacturers" (level-2 suppliers). Note that this problem, which can be generalised to K levels, is *much* harder than the classical generalised assignment problem, as in general it cannot be broken up into multiple problems to solve separately.

We generalised the GA to handle a 3-layer problem by introducing a second chromosome, of length N_2, containing information concerning the assignments between warehouses and manufacturers. This information is stored so that i^{th} entry of the chromosome represents the index of the manufacturer to whom warehouse i sends its orders. The GA then works basically as in the simpler 2-layer case, apart from some minor modification to the genetic operators (e.g. crossover, occurring with some probability p_c, mixes either the first chromosomes of the two individuals or the second chromosomes, with equal probability).

As described by Equation (2), the demand $x_i^{[2]}$ of each warehouse i is retrieved from the orders of the retailers. Together with the fixed supply of the manufacturers this creates the constraint in Equation (3). As in the simpler 2-layer case, these constraints are handled by using an unfitness measure, with a little modification:

$$u_l(E) = \sum_{h=1}^{N_3} \Big[\max \left(0,\ \sum_{i=1}^{N_2} \theta_{ih} x_i^{[2]} - s_h \right) \ + \ \sum_{i=1}^{N_2} (1 - \theta_{ih}) x_j^{[2]} \Big]$$

where s_h, here, is the fixed supplies of the manufacturers. The first term in Eq. 2 is the sum of all the requests which exceed the fixed supply of each manufacturer, whereas the second term is the sum of all orders placed by retailers to non-supplied warehouses. This measure is therefore zero if and only if no manufacturer has more requests than supply and all warehouses with requests are served by a manufacturer.

We implemented these extensions to the original GA and ran it for many different maps and with many different parameter sets, verifying that it can effectively be used as benchmark for the agent-based models soon to be described.

3 Agent-based models

3.1 A "reinforcement" agent-based model

The first model is what we call a "reinforcement" agent-based model for a 3-layer supply chain model. As shown in Figure 3 the facilities are now agents with demand and supply. Each individual agent tries to fulfill his demands without global knowledge.

This is a process divided into three different phases, the requesting phase, the shipping phase and the reinforcement phase. During the first phase the retailers start with demanding goods from the warehouses, which then request goods from the manufactures. In the second phase the goods are shipped in the opposite direction. After the shipping phase the connections between the nodes get reinforced, hence the name of the model. With these reinforced connections the process returns to the first phase.

Now we take a closer look at the requesting phase. Each agent , i, demands goods from the higher layer, j, based on probabilities P_{ij}, calculated as:

$$P_{ij} = F_{ij}^{\beta} \cdot \eta_{ij}^{(2-\beta)}, \tag{5}$$

where F_{ij} is the fidelity matrix, η_{ij} is the visibility matrix and β is a relative weighting factor for the two matrices. The fidelity matrix acts like a memory of the previous iterations through the reinforcement processes and the visibility matrix has the inverse distances, w_{ij} as entries. The visibility matrix also stays unchanged throughout the whole process. With such a definition of the visibility matrix we ensure that a close store is more probable then one further away. This should lead to an optimisation of the profit. The

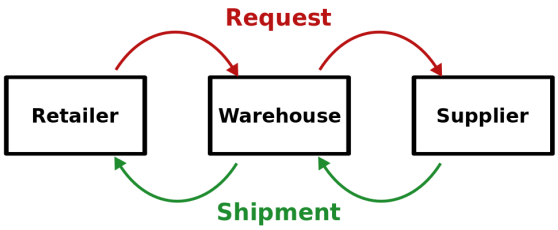

Figure 3: *Interaction scheme of the three layers, where the facilities are agents with demand and supply, the red arrows represent the requesting phase while the green arrows the shipment phase.*

rows of the probability matrix are then normalised and a shop will be probabilistically selected with this probability matrix. After the retailers gone through this process the warehouses request goods from the manufacturers in the same manner.

When the goods have been requested the shipping phase begins, where the manufacturers have to distribute the goods. This is done by selecting the customers probabilistically, where the probability matrix P_{ij} is reused. Here we use the columns of P_{ij} instead of the rows and normalise them afterwards. With this selection rule the strength of a bond between a store and its customers determines the selection. The supplies of the customers are updated, where we assumed that all orders from one customer are shipped at once. We also implemented another constraint to prevent unprofitable transactions, if the profit is smaller than zero we deny the shipment. The profit is calculated similarly as in Equation 1 with the difference that we only look at one agent and if its trade is profitable. For this we added another argument, the shipment distance, for the agents. The shipment distance D is initialised with zero for the manufacturers and will then be recursively increased from layer to layer. This means that the retailers shipment distance is the sum of the distances between him and the warehouse, $w_{ij}^{[1]}$, and between the warehouse and the manufacturer, $w_{ij}^{[2]}$. With this new argument we can write the profit f_i for an agent i as:

$$f_i = V_i \cdot (1 - \alpha \cdot D_i), \tag{6}$$

where V_i stands for the trade volume. If the profit is larger then zero we allow the shipment and the customer get his items shipped. The trade is completed when the retailers receive the shipments.

Finally, in the last phase, we have to evaluate the success of the shipments to reinforce the connections. To do so we check if the customer received his order from the store. We do not have to check for profitability as there is no shipment if it is unprofitable. If the customer was satisfied we reinforce the entry in the fidelity matrix between customer i and store j:

$$F_{ij} = F_{ij} + \delta, \tag{7}$$

where δ is a defined gain constant. As we want to have a convergence solution we do allow a exploration phase in the beginning and later we force the system to converge. We can reach this if we use a decay mechanism with decay constant ρ in the fidelity matrix:

$$F_{ij} = (1 - \rho)F_{ij}. \tag{8}$$

3.2 A "market-oriented" agent-based model

The main idea of the agent-based approach is to let the individual nodes, or agents, perform transactions based on prices, and continually update the price value at each node. The prices determine the probabilities of different transactions to be performed. The transactions between the nodes generates values of supply and demand at each node, which in turn determine how the prices are updated. And so prices converge over time to generate effective solutions of the problem of generating the flows in the supply chain network.

The algorithm runs in to steps, the trade-step and the price-update-step. In the trade-step, two layers at a time interact with each other, where one layer act as "customers" and the other layer acts as "stores". Take for example in a 3-layer supply chain with retailers, warehouses and manufacturers. In the trade-step in this case, retailers, in the role as "customers", will first interact with warehouses, in the role as "stores". Then warehouses will change its role to "customers" and will interact with manufacturers in the role as "stores".

Each node in each layer will have four scalar values; demand, supply, price and location. In the interaction between two layers in the trade-step, an matrix $P(i, j)$ will be generated through

$$P_{ij} = \max\{0, f(\text{customerPrice}_i - \text{storePrice}_j - t_c d_{ij})\} \ , \tag{9}$$

where d_{ij} is the distance from customer i to store j and t_c is the transport cost per unit distance. f is some function, for example $f(x) = x$ in the simples case. The matrix P is normalized and used to randomly select customer-store-pairs (i, j) to perform transactions. For each transaction, the demand value of customer node i and supply value of store node j are incremented. When the demand of a customer node has been satisfied, all the values of row i in the probability matrix P are set to zero. When there are no non-zero elements in P there are no more possible transactions, and the interaction between the two layers are completed.

When all the trade sequences have been performed between the layers, the price-update-step is performed. There are many methods to update the prices to be explored. In the simplest case, prices are updated as follows:

$$\text{price}_i \leftarrow \text{price}_i + k(\text{demand}_i - \text{supply}_i) \ , \tag{10}$$

where k is a set parameter. Through exploration of different values, $k = 0.01$ was found to be effective.

The convergence of prices in the algorithm is a central desirable feature. If the algorithm is successful, the prices will converge to direct an effective flow of products through the network. The mechanism that is constructed to update the prices, and the way that the probabilities are generated in the trade-step, will determine how fast the prices will converge, if at all. And also the quality of the solutions that are generated.

4 Results and discussion

4.1 Comparison of the models

In Figures 4, 5 and 6, we present a comparison among the agent-based models described in the previous section and the GA, using the same map with 45 retailers (blue circles), 5 warehouses (green squares) and 3 manufacturers (red triangles) placed in a 1×1 square. In all three cases the transportation cost was set to $\alpha = 0.5$. In the left panels, the profit of the supply chain over time is displayed, whereas in the right panels the final network configuration found in the three cases is shown. The flow of items from manufacturers to warehouses is represented by red edges of proportional width.

A comparison of the performance of the three methods is also shown in Figure 7. Histogram bars represent the average profit of the supply chain over 20 different maps with the same number of facilities, with standard deviations as error bars. This comparison shows that the reinforcement model finds on average a network yielding a profit comparable to that found by the GA. The performance of the market model, instead, is on average a bit poorer, with much larger variation in different maps.

4.2 The effect of parameter α

Some simulations exploring the effect of parameter α on the best network found by the GA and on the behaviour of the AB models revealed what might be intuitively expected. If the transportation cost of one unit of product per unit distance is too high, shipping products to customers located too far away from a supplier is unprofitable, and therefore more and more isolated nodes appear. This parameter also seems to play a role in the convergence speed of the GA, although this has not been investigated in depth as we did not think it was too interesting.

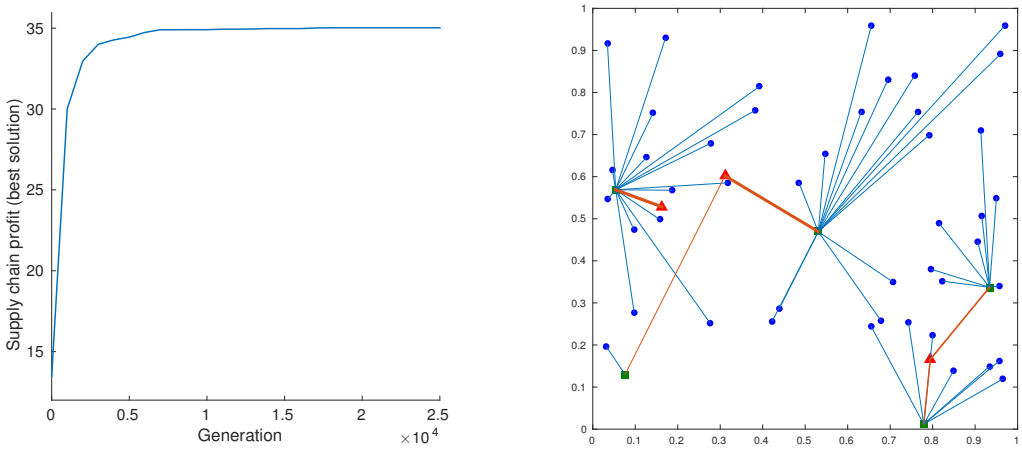

Figure 4: *3-layer supply chain optimized with the GA.*

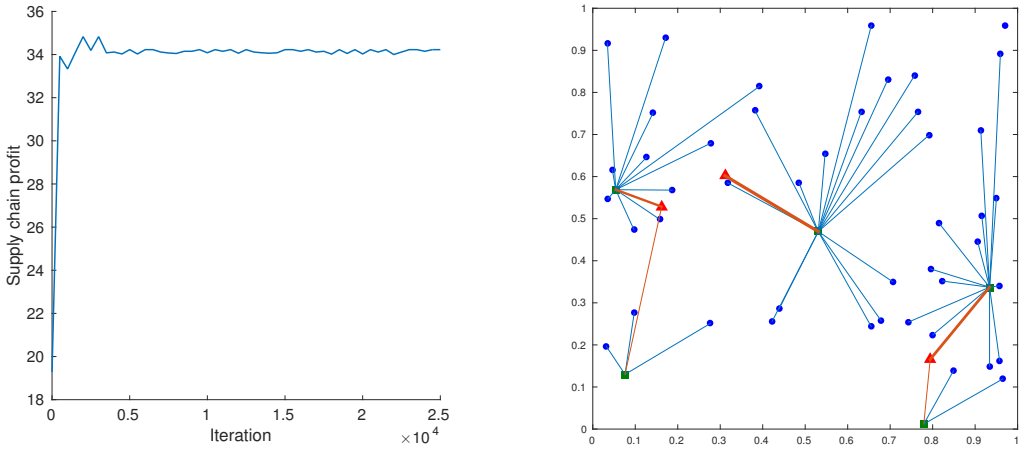

Figure 5: *Evolution of the "reinforcement" agent-based model for a 3-layer supply chain.*

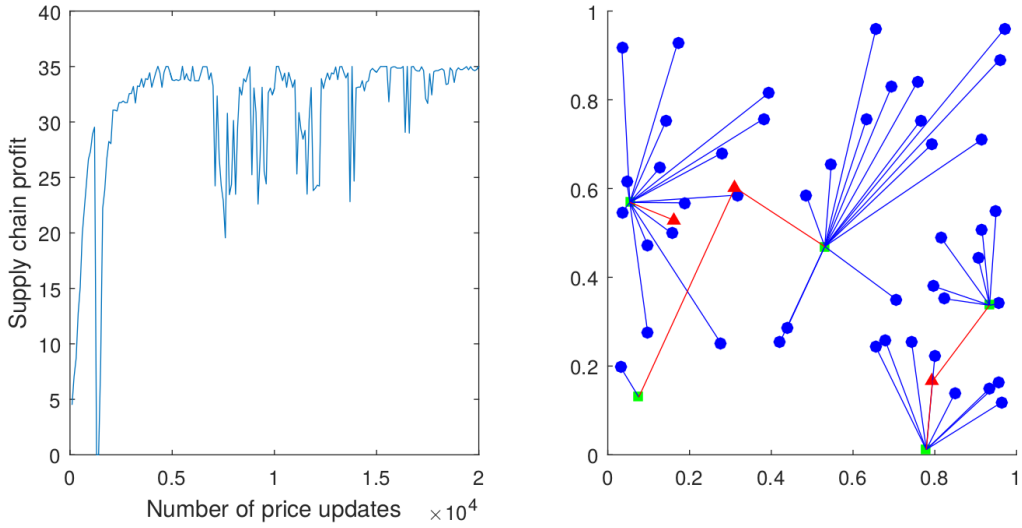

Figure 6: *Evolution of the "market-oriented" agent-based model for a 3-layer supply chain.*

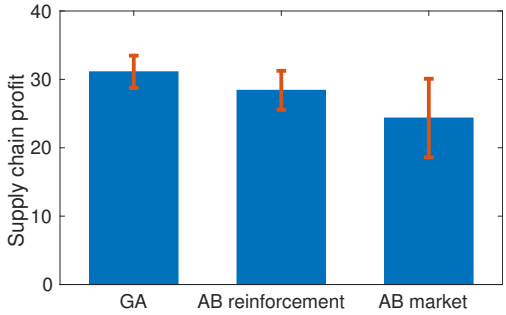

Figure 7: *Comparison of the performance of the three methods, with best parameter sets found. Displayed points are averages over 20 random maps, with standard deviations.*

For these reasons, we decided to keep α fixed to 0.5 in our analyses, a value that seemed reasonable in that it makes a map "solvable", but not too easy, i.e. a positive profit is only achieved if a good network configuration is found, and this generally corresponds to a large proportion of customers being satisfied.

4.3 The reinforcement agent-based model and its dependence on parameters

As shown in Figure 7, the reinforcement agent-based model works relatively well compared to the genetic algorithm, in that it successfully converges to a highly profitable steady state. The data in Figure 7 was obtained by simulating all models on 20 common maps, with "optimal" parameters. By this, we mean that each model was first explored individually to find the parameter set yielding the desired behaviour.

The reinforcement model, in particular, depends on three parameters: β, controlling the relative importance of fidelity over visibility, the reinforcement constant δ and the decay rate ρ. Some investigation revealed that the model works best if δ and ρ are set to the same (or very similar) value. This may seem counter-intuitive, since the former is an additive gain and the latter is an exponential decay rate. However, a "good" reinforcement behaviour is only achieved when they are of the same order of magnitude, and we believe an analytical investigation might shed some light on this.

As for parameter β, the performance dependence is shown in Figure 8. On average, the model converges to a highly profitable configuration for β values around $1 \sim 1.5$, i.e. corresponding to a good balance between fidelity and visibility. The reasons for this can be probably understood if we recall Ant Colony methods, in which a similar trade-off is also needed for optimal performance.

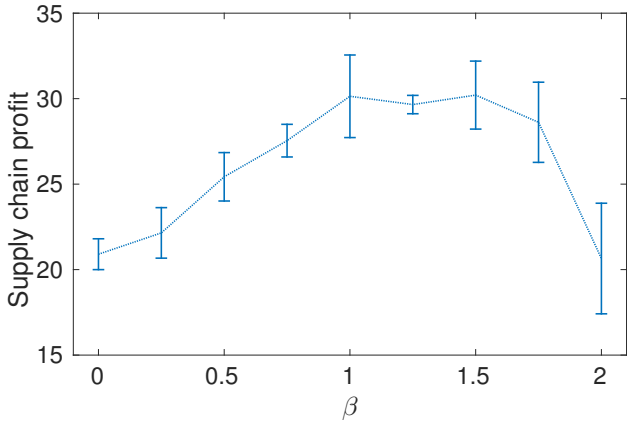

Figure 8: *Performance of the reinforcement AB model as a function of parameter β. Averages over 5 maps, with standard deviations.*

4.4 Market-oriented agent-based model

Some desirable properties of our constructed methods are speed, convergence, high fitness values and consistence of results over different maps and runs. As we can see in figure 7 the market-oriented method was the least reliable. A problem with the algorithm was that the prices did not converge completely over time. The price at every node approached some value, around which the prices oscillated. There is an explanation for this. The dynamics of the price for each agent can be described by

$$\Delta P = k(D(P) - S(P)) \tag{11}$$

The price is therefore stable at the equilibrium price-point where supply equals demand. In our case, supply and demand as functions of price are step-functions, and $\Delta P = 0$ generally lacks a solution. As the prices in the system never settles, the volumes of product flows never settles either. As a result the fitness values of the generated solutions fluctuates even after approximately "good" (in terms of what the system strives towards) prices has been reached in the system. This behaviour is clearly visible in figure 6. At first prices are adjusted and tuned, and the profit of the solutions increase quickly. The profit then hovers close to the optimal solution, with minor and occasionally significant deviations.

One simple modification to solve this problem would be to use elitism. That is save the solution with the highest fitness value so far. This is an obvious measure that would neglect all noise in the system below that is generating the solutions, as long as a few good solutions are generated. It is still interesting to make the prices and the solutions generated to converge. One approach to solve this problem could be to modify the algorithm to use continuous values for product volumes when updating supply and demand, rather than discrete. Then It would be possible for prices to converge.

Additional ideas for modification of the algorithm would be to introduce use of control theory. As described in (11) the price delta for an agent is proportional to the input signal $u = D(P) - S(P)$. The price updates could just as well take into account the rate of change of the input signal, and the sum of the input signal over time. As in a PID-controller. Further exploration could then be made to find proper parameters k_P, k_I and K_D.

5 Conclusions

What we discovered was that different values of α seemed to effect the design of the flow in the network a lot. As the value of α increased the amount of customers not receiving any shipment increased and therefore decreased the customer satisfaction. We also discovered how the parameter β affected the total profit of the network and that the optimal β was the one with a good balance between fidelity and visibility.

From our methods we found out that the genetic algorithm, as expected, constructed flows with higher profit than the agent-based models in average. There was however exceptions for a few maps where the agent-based received higher profit. For the other maps the outcome of the agent-based methods still came fairly close to the outcome of the GA. Between our agent-based methods the one receiving the highest average profit was the one with reinforcement although a few maps favoured the market-oriented model. The model with the poorest results was the market-oriented model and we realised that the reason was the discreteness in the model.

References

[1] Fulya Altiparmak et al. "A genetic algorithm approach for multi-objective optimization of supply chain networks". In: *Computers & Industrial Engineering* 51.1 (2006), pp. 196–215.

[2] P.C. Chu and J.E. Beasley. "A genetic algorithm for the generalised assignment problem". In: *Computers & Operations Research* 24.1 (1997), pp. 17–23. ISSN: 0305-0548. DOI: http://dx.doi.org/10.1016/S0305-0548(96)00032-9. URL: http://www.sciencedirect.com/science/article/pii/S0305054896000329.

[3] Ram Ganeshan and Terry P Harrison. "An introduction to supply chain management". In: *Department of Management Science and Information Systems, Penn State University* (1995).

[4] Admi Syarif, YoungSu Yun, and Mitsuo Gen. "Study on multi-stage logistic chain network: a spanning tree-based genetic algorithm approach". In: *Computers & Industrial Engineering* 43.1 (2002), pp. 299–314.

[5] Caroline Thierry, Gérard Bel, and André Thomas. "Supply chain management simulation: an overview". In: *Simulation For Supply Chain Management* (2008), pp. 1–36.