Genetic algorithm for inventory positioning problem with general acyclic supply chain networks

Article *in* European J of Industrial Engineering · January 2016

DOI: 10.1504/EJIE.2016.076385

4 authors, including:

Haitao Li
University of Missouri - St. Louis
39 PUBLICATIONS **415** CITATIONS

SEE PROFILE

Genetic algorithm for inventory positioning problem with general acyclic supply chain networks

Dali Jiang

Institute of Modern Logistics,
Logistical Engineering University,
Chongqing 401311, China
Email: jiang2100-2@163.com

Haitao Li*

School of Economics,
Tianjin University of Commerce,
Tianjin 300134, China
and
Department of Logistics and Operations Management,
College of Business Administration,
University of Missouri – St. Louis,
One University Blvd, St. Louis, MO 63121, USA
Email: lihait@umsl.edu
*Corresponding author

Tinghong Yang and De Li

Institute of Modern Logistics,
Logistical Engineering University,
Chongqing 401311, China
Email: yth_ah@163.com
Email: lide_whcq@hotmail.com

Abstract: Inventory positioning, also known as safety stock placement, in supply chain networks is an important optimisation problem that has various applications in supply chain design and configuration. In this paper, we develop a new genetic algorithm (GA) for this NP-hard problem. Our new GA features custom designed procedure to generate feasible individuals by exploiting the problem structure. It also implements a multi-start strategy to enhance solution quality. Computational results show that our GA is able to offer near optimal solutions in reasonable computational time. [Received 4 October 2014; Revised 19 October 2015; Revised 14 January 2016; Accepted 18 January 2016]

Keywords: inventory positioning; safety stock placement; general acyclic network; genetic algorithm; GA; supply chain design.

Reference to this paper should be made as follows: Jiang, D., Li, H., Yang, T. and Li, D. (2016) 'Genetic algorithm for inventory positioning problem with general acyclic supply chain networks', *European J. Industrial Engineering*, Vol. 10, No. 3, pp.367–384.

Biographical notes: Dali Jiang received his Masters in Logistics Engineering in 1995 at Logistical Engineering University in Chongqing, China, and PhD in Transportation and Logistics Management in 1998 from Southwest Jiaotong University, Chengdu, China. Since 1992, he is a faculty member of Logistical Engineering University, currently working as Professor and Director of the Institute of Modern Logistics, Logistical Engineering University. During the period of March 2009 to March 2010, he worked as a Visiting Scholar in University of Missouri – St. Louis, USA. His research interests are in logistics and supply chain management. He is the executive director of China Society of Logistics (CSL).

Haitao Li is Associate Professor of Logistics and Operations Management and Research Fellow of Centre for Transportation Studies, at University of Missouri – St. Louis, USA. His research interests include optimisation modelling, simulation, and algorithm design in the application domains of supply chain network design, project scheduling and workforce optimisation. His research has been published in scholarly journals including *European Journal of Operational Research*, *Computers and OR, Omega*, *Military Operations Research*, *Journal of Scheduling*, *Interfaces* and *Annals of Operations Research* among others.

Tinghong Yang is an Associate Professor for Applied Mathematics at Logistical Engineering University in Chongqing, China. Currently, he is also a PhD student in Logistics Management. His research interests are supply chain management and logistics modelling. He is a Principal Investigator on several projects funded by the National Natural Science Foundation of China.

De Li received his PhD in Logistics Management in 2013 at Logistical Engineering University, Chongqing, China. Currently, he is a Lecturer in the Department of Management Science and Engineering at Chongqing University. His research interests are in supply chain management, warehouse modelling and virtual logistics.

1 Introduction

An inventory positioning problem aims to optimise the location and level of inventory throughout a supply chain network. Given a network of supply chain entities, i.e., parts, components, semi-finished goods and finished goods, the goal is to minimise the total system wise inventory cost while satisfying certain service guarantee for customers (Minner, 2000). The problem with special network structures such as a serial and spanning tree structure can be efficiently solved by dynamic programming (Minner, 1997; Graves and Willems, 2000). When the underlying supply chain network has a general acyclic structure, the problem becomes NP-hard (Sahni, 1974).

Various computational algorithms have been developed for solving the inventory positioning problem with general acyclic structure. Notably, Magnanti et al. (2006) developed an exact piecewise linear approximation approach that dynamically generates linear pieces to approximate the concave objective function. Shu and Karimi (2009) proposed an iterative linear programming heuristic to obtain near optimal solutions efficiently. Li and Jiang (2012) developed new framework based on project scheduling to model the problem, and employed constraint programming (CP), a methodology in

artificial intelligence, as the solution method. To enhance the solution quality, they improve the solutions obtained by CP using a genetic algorithm (GA). The hybrid CP-GA algorithm performs well for the benchmark instances with size up to 80 nodes. Humair and Willems (2011) developed an exact algorithm for solving the problem with general acyclic network and non-concave inventory cost function of each stage's inbound and outbound service times, but with no capacity constraint. They also devised two heuristics, and tested their algorithms on the real world instances of Willems (2008). Recent work by Grahl et al. (2014) relaxed the assumption of identical service times quoted to successors by considering differentiated service times. They developed a local search heuristic, GA and simulated annealing (SA) metaheuristics to solve the addressed problem, and showed that SA outperforms other heuristics. None of the exact and heuristic methods in Humair and Willems (2011) and Grahl et al. (2014) considers the capacity limits on safety stock level at each node.

The effectiveness of the CP-GA algorithm in Li and Jiang (2012) depends on the condition that the CP algorithm is able to find a sufficient number of solutions, which form an initial population for GA. In some situations, the CP approach may not be able to generate sufficient number of feasible solutions, which may affect the performance of GA. The purpose of this paper is to present a new GA algorithm to enhance the solution quality. Comparing with the GA procedure embedded in the CP-GA algorithm of Li and Jiang (2012), several advancement will be made on the approach to generate the initial population and genetic operators. The algorithm was tested on the benchmark instances in the literature.

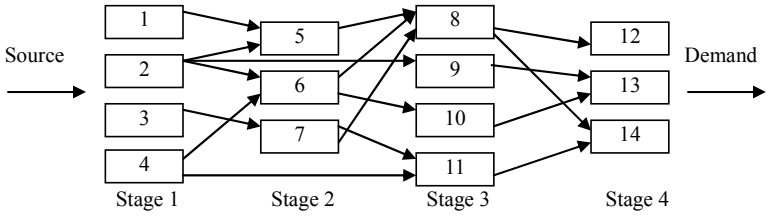
The remainder of the paper is organised as follows. Section 2 formally describes the inventory positioning problem in general acyclic networks and its mathematical formulation. In Section 3, we describe the new GA algorithm including the encoding scheme, procedure to generate feasible solution, GA operators and the overall algorithm framework. Computational results are presented in Section 4. Section 5 draws conclusions and discusses future research directions.

2 Problem description and model formulation

Consider a general acyclic supply chain network $G(N, A)$, where N is the node set representing supply chain entities of sourcing, procurement, production, and assembly stages, and A is the arc set denoting the demand dependencies and temporal relationships among the entities. The network is spanning tree when the number of arcs equals the number of nodes minus one, i.e., $|A| = |N| - 1$. A general acyclic network includes spanning tree as a special case, i.e. it is possible that $|A| \geq |N| - 1$, but no cycle or loop is allowed.

A generic example of acyclic supply chain network is illustrated in Figure 1. The network has four stages and 14 nodes (entities), where stage 1 consists of four sourcing nodes from external suppliers; stage 2 includes three manufacturing nodes; stage 3 has four assembly nodes; and stage 4 represents three finished goods. The higher level of safety stock kept at each node, the less chance for a node to experience stock out, i.e. the higher the service level will be. However, such higher safety stock level will incur higher inventory holding cost.

Figure 1 An example of general acyclic supply chain network.



Node j is assumed to have a constant lead time T_j and a safety stock holding cost rate h_j per item per time unit. For an arc (i, j), it is required that node j must not start its inbound service time before the outbound service time of node i. There is a capacity limit of C_j time periods of safety stock for each node j. Finished products have to be delivered within some customer-specified deadline d_j on time. The problem is to determine the level of safety stock at each node to minimise the total inventory holding cost of the supply chain network.

Define SI_j as the decision variable of inbound service time of node j, and S_j as the outbound service time. Both SI_j and S_j are integer, which is in line with the fundamental periodic review policy assumed for each j. Because the safety stock level in terms of days at j is a function of the net replenishment time, i.e., $SI_j + T_j - S_j$, we seek to find optimal inbound and outbound service times of each node that minimise the total inventory holding costs. Following Magnanti (2006), Shu and Karimi (2009) and Li and Jiang (2012), the mathematical programming formulation for the capacitated inventory positioning problem in general acyclic network can be written as:

$$\min \sum_{j \in N} h_j \Phi_j \left(SI_j + T_j - S_j \right) \tag{1}$$

s.t.

$$S_j \le d_j \qquad \forall j \in D \tag{2}$$

$$S_j - SI_j \le T_j \tag{3}$$

$$SI_j + T_j - S_j \le C_j \tag{4}$$

$$SI_j - S_i \ge 0 \qquad \forall (i, j) \in A \tag{5}$$

$$SI_j, S_j \in Z^+ \qquad \forall j \in N \tag{6}$$

The objective function (1) minimises the total safety stock cost of all nodes $j \in N$. $\Phi_j(\cdot)$ denotes the amount of safety stock at node j, which is a non-decreasing and concave function of the net replenishment time $SI_j + T_j - S_j$ constraint (2) guarantees that the outbound service time of each finish good node cannot exceed the customer quoted delivery time d_j, where D is the set of nodes representing all finished goods. When the delivery time d_j is tight, the outbound service time S_j will be forced to be small, which will result in a larger net replenishment time and higher safety stock level and cost. Constraint (3) ensures that the net replenishment time is non-negative. Constraint (4) states that the net replenishment time of each node cannot exceed some limit, which imposes a capacity constraint on the level of safety stock at each node. Constraint (5)

makes sure that for each arc (i, j), the inbound service time of j cannot exceed the outbound service time of its predecessor i. The integer requirement of all decision variables is specified in (6). The problem described by (1)–(6) minimises a concave objective function, which is well-known to be NP-hard (Sahni, 1974).

3 GA algorithm

GA is a random search technique that mimics the phenomenon of natural evolution (Goldberg, 1989; Davis, 1991). GAs do not use any information on differentiability, convexity or other auxiliary characteristics, which makes them suitable for a variety of optimisation problems in logistics and supply chain, e.g., capacitated plant location (Gen et al., 1999), fixed charge network design (Jaramillo et al., 2002), minimum spanning tree (Zhou and Gen, 1999), network design (Palmer and Kershenbaum, 1995) and warehouse allocation (Zhou et al., 2003). One distinctive advantage of GA is its convenience for handling nonlinear optimisation problems, which are often encountered in supply chain applications.

In this section, we present a new GA algorithm for solving the nonlinear optimisation problem (1) to (6). It includes two main components. The first component is a procedure to generate initial population of feasible solutions by employing mathematical deduction and randomisation, which exploits the structure of the problem at hand. The second component is a GA framework to improve the initial population through evolution of chromosomes.

3.1 Encoding scheme

Success of GA relies heavily on an effective encoding scheme to represent a solution to the problem at hand. For the 0–1 binary programs, the binary string encoding is a straightforward choice (Goldberg, 1989; Davis, 1991, Gen et al., 1999; Jaramillo et al., 2002; Zhou and Gen, 1999; Palmer and Kershenbaum, 1995; Zhou et al., 2003; Goldberg and Lingle, 1985). For our problem with general integer decision variables, the binary coding scheme is also applicable (Orero and Irving, 1997; Pruettha and Konlakarn, 2001; Yokota et al., 1996), which usually works well for small scale problems. When the problem size and scale is large, such binary coding scheme becomes inefficient due to the following facts:

1 the search space will become exponentially large

2 the conversion between binary strings and integers costs extra computational time

3 the efficiency of the binary encoding may be lost when converting back and forth between binary and integer values (Hung and Chen, 2006).

To circumvent the above difficulties of binary encoding, researchers have proposed various encoding schemes. Hua and Huang (2006) proposed a variable-grouping based GA, in which decision variables are coded with continuous relaxation, and then grouped appropriately to reduce the search space. Others employ integer encoding methods. Notably, Harper et al. (2005) used an ordered structure (s-dimensional vector) of integers to represent the solution space. Moon et al. (2008) developed a combined structure of

assignment indexes and processing orders to construct the chromosome. Chung et al. (2009) employed each gene of the chromosome to represent a train, which is assigned to a route represented by its allele. These methods share the similar idea as proposed by Hadj-Alouane and Bean (1997), that the chromosome represents a solution with random numbers in a specific ordered structure so that the size of the search space can be reduced sharply. Despite the successful application of the above integer encoding methods, they may lead to a very large search space given the size and scale of our optimisation problem at hand. Therefore, we have developed a novel direct integer encoding scheme for our GA.

Each chromosome is based one-dimensional array which consists of positive integer values, directly representing decision variables S_j (j = 1, 2,…,n), which is the outbound service time that each node j promises to its downstream customers. Note that the inbound service times SI_j (j = 1, 2,…,n) can be determined by $SI_j = \max_{(i,j)\in A} S_i$.

The advantage of such direct integer encoding is the significant reduction of search space, comparing with the binary encoding schemed reviewed earlier. However, the trade-off is the lack of well-developed genetic operators reported in the literature. In the following sub-sections, we present customised procedures and operators based on our encoding scheme.

Figure 2 Representation of a chromosome

S_1	S_2	……	S_n

3.2 Procedure to generate an individual

We take the following steps to get a random feasible solution for generating an individual. The values of S_j and SI_j of node j are determined according to the reversed order of stages. That is, to determine the demand nodes first, then the predecessor nodes of demand nodes, until all the source nodes are determined.

Phase I: Determine the feasible interval of SI and S for all nodes.

Step I.1 Find feasible interval $\left[msi_j^1, MSI_j^1\right]$, $\left[ms_j^1, MS_j^1\right]$ by the quoted service time of demand nodes.

I.1.1 $\forall j \in N, BZ_j = 0$;

I.1.2 $\forall j \in D,\ ms_j^1 = d_j, MS_j^1 = d_j, BZ_j = 1$;

$$msi_j^1 = \max\{ms_j^1 - T_j, 0\}, MSI_j^1 = \max\{MS_j^1 - T_j + C_j, 0\}$$

where T_j is the lead time of node j, and C_j is the safety stock days for demand node j.

Find a node j whose (BZ_j == 0) and ($\forall(j, k) \in A\ BZ_k$ == 1), and let

I.1.3 $$msi_j^1 = \max\{ms_j^1 - T_j, 0\}, MSI_j^1 = \max\{MS_j^1 - T_j + C_j, 0\}$$

I.1.4 If $\forall j \in N, BZ_j$ == 1 then terminate, else go to step 3;

Step I.2 Determine the feasible interval $[msi_j^2, MSI_j^2], [ms_j^2, MS_j^2]$ by $SI = 0$ for all

supply nodes.

I.2.1 $\forall j \in N$, $BZ_j = 0$.

I.2.2 Find the set SP of all supply source nodes, and let $\forall j \in SP$,

$$msi_j^2 = 0,\ MSI_j^2 = 0,\ ms_j^2 = \max\{msi_j^2 + T_j - C_j, 0\},$$

$$MS_j^2 = MSI_j^2 + T_j$$

I.2.3 Find any node j whose ($BZ_j == 0$) and ($\forall (i,j) \in A\ BZ_i == 1$), and let

$$msi_j^2 = \max_{(i,j)\in A} msi_i^2,\ MSI_j^2 = \max_{(i,j)\in A} MS_i^2$$

$$ms_j^2 = \max\{msi_j^2 + T_j - C_j, 0\},\ MS_j^2 = MSI_j^2 + T_j$$

I.2.4 If $\forall j \in N$, $BZ_j == 1$ then terminate, else go to step 7.

At last, the feasible interval of S and SI can be got by the intersection of both.

$$\forall j \in N,\ msi_j = \max\{msi_j^1, msi_j^2\},\ MSI_j = \min\{MSI_j^1, MSI_j^2\}$$

$$ms_j = \max\{ms_j^1, ms_j^2\},\ MS_j = \min\{MS_j^1, MS_j^2\}$$

Phase II: Determine S_j and SI_j for the demand nodes

Step II.1 Find a demand node j, set its S_j equal to its deadline d_j:

$$S_j = d_j, \tag{7}$$

Step II.2 Set SI_j of demand node j as a random integer by the following formula:

$$SI_j = msi_j + \lfloor r_j * (MSI_j - msi_j) \rfloor, \tag{8}$$

where r_j is a real number generated randomly in [0, 1]. From equation (8), we can get inequality (9), which ensures that the obtained values of S_j, SI_j satisfy constraints (3) and (4).

$$\max\{0, S_j - T_j\} \le SI_j \le S_j - T_j + C_j \tag{9}$$

Step II.3 Repeat step 9 and 10 until demand nodes all have been set.

Phase III: Determine S_j and SI_j of nodes whose successor nodes all have fixed S and SI

Step III.1 Find a node i, which has its all successor nodes j with the determined values of S_j, SI_j. Set S_i by the following formula.

$$S_i = \min_{(i,j)\in A} SI_j \tag{10}$$

The produced integer S_i clearly satisfies constraint inequality (5).

Step III.2 Set SI_i of node i as a random integer by the following formula:

$$SI_i = \max\left\{0, \min\left\{MS_i - T_i, \lfloor S_i - T_i + r_i * C_i \rfloor\right\}\right\}, \tag{11}$$

From equation (11), we can get inequality (12), which satisfies constraints (3) and (4).

$$\max\left\{0, S_i - T_i\right\} \le SI_i \le S_i - T_i + C_i \tag{12}$$

Step III.3 Repeat step III.1 and III.2 until all nodes have been determined.

Phase IV: Determine S_j and SI_j for the supply nodes

Step IV.1 Find all supply nodes, set SI_j equal to 0 and set $S_j = T_j$ if $SI_j + T_j - S_j < 0$ for all supply node j.

Step IV.2 Set $SI_i = \max\limits_{(k,i)\in A} S_k$ for all node i.

Step IV.3 Set $S_i = SI_i + T_i$ if $SI_i + T_i - S_i < 0$ for all node i.

Step IV.4 Repeat step IV.2 and IV.3 until nothing have been changed.

After all the nodes in an acyclic capacitated supply chain network have been determined with their values of S_j, SI_j through the above procedure, a feasible solution to the acyclic capacitated safety stock placement problem is obtained. We then set all the alleles of a chromosome with the obtained S_j to generate a feasible individual.

3.3 Genetic operators

3.3.1 Parent selection operator

Parent selection is an important process that guides a GA search toward promising regions in a search space. There are a number of different selection methods, such as roulette wheel selection, tournament selection, rank selection, elitism selection, and random selection (Gen and Cheng, 2000). For our algorithm, we choose the roulette wheel selection method to keep the diversity of chromosomes.

3.3.2 Crossover operator

The crossover operator generates new children by combining information contained in the chromosomes of the parents, so that the new chromosomes will have the best features of their parents' chromosomes. There are several types of crossovers, including single-point crossover, multi-point crossover, and uniform crossover (Gen and Cheng, 2000). In order that the information of parents with integer encoding can be passed on definitely, we create children by taking a weighted average of the parents. We adopt an adjusted arithmetic crossover operator, letting

$$Child = \lfloor parent1 + r(parent2 - parent1) \rfloor = \lfloor (1-r)parent1 + rparent2) \rfloor \tag{13}$$

where r is a random real number in [0, 1]. We are able to prove that if the parents are feasible, children produced by this kind of crossover are always feasible (Li and Jiang, 2012).

3.3.3 Mutation operator

The rationale of mutation is to provide sufficient amount of randomness to explore less visited solution space and prevent premature convergence. In our GA, the mutation operator randomly selects one gene of a chromosome, and randomly selects a mutation node r. The effect of mutation node inventory change could be two aspects: on one hand, inbound service time SI_r may be changed, so S and SI of its upstream nodes should be modified by constraints (3) and (4). On the other hand, outbound service time S_r may be changed, so SI and S of its downstream nodes should be modified by constraints (3) and (4). The mutation operations include both the SI_r mutation operation and S_r mutation operation. If the mutation node r belongs to demand nodes then mutate its SI_r; if the mutation node r is belongs to supply nodes then mutate its S_r; if the mutation node r does not belong to demand nodes or supply nodes then randomly mutate its SI_r or S_r with equal probability.

1 Determine SI for all nodes

Step 1 Find a supply node j, set its SI_j equal to zero:

$$SI_j = 0,$$

Step 2 Find a node j, which has its all predecessor nodes with the determined values of SI and S. Set SI_j by the following formula.

$$SI_j = \max_{(i,j)\in A} S_i$$

Step 3 Repeat Steps 1 and 2 until SI all have been set.

2 Mutate SI_r

Replaces the allele with a random integer using equation (8), let $SI_r = msi_r + rand * (MSI_r - msi_r)$. If $SI_r + T_r - S_r < 0$ then set $S_r = SI_r + T$; if $SI_r + T_r - S_r > C$ then set $S_r = SI_r + T_r - C_r$. And modify S_j and SI_j of its downstream nodes and upstream nodes by a recursive routine.

Phase I: Modify S_j and SI_j of downstream nodes

I.1 Set $SI_j = \max_{(i,j)\in A} S_i$ for all downstream node j of r; if $SI_j + T_j - S_j < 0$ then set $S_j = SI_j + T_j$; if $SI_j + T_j - S_j > C_j$ then set $S_j = SI_j + T_j - C_j$.

I.2 Let r be one successor node of node j and set $SI_r = \max_{(k,r)\in A} S_r$. If $SI_r + T_r - S_r < 0$ then set $S_r = SI_r + T_r$; if $SI_r + T_r - S_r > C_r$ then set $S_r = SI_r + T_r - C_r$.

I.3 Repeat recursively step I.1and I.2until no more change is needed.

It is important that the mutation is infeasible if any S_j of demand node j has been modified to be $S_j > d_j$.

Phase II: Modify S_j and SI_j of upstream nodes

II.1 Set $S_j = \min_{(j,i)\in A} SI_i$ for all upstream node j of r; If $SI_j + T_j - S_j < 0$ then set $SI_j = S_j - T_j$; if $SI_j + T_j - S_j > C_j$ then set $SI_j = S_j - T_j + C_j$.

II.2 Let r be one predecessor node of node j and set $S_r = \min_{(r,k)\in A} SI_k$. If $SI_r + T_r - S_r < 0$ then set $SI_r = S_r - T_r$; if $SI_r + \underline{T_r} - S_r > C_r$ then set $SI_r = S_r - T_r + C_r$.

II.3 Repeat recursively Step II.2 and II.3 until nothing have been changed.

3 Mutate S_r

Replaces the allele with a random integer using equation $S_r = ms_r + \lfloor r*(MS_r - ms_r) \rfloor$. If $SI_r + T_r - S_r < 0$ then set $SI_r = S_r - T_r$; if $SI_r + T_r - S_r > C_r$ then set $SI_r = S_r - T_r + C_r$. And modify the S_j and SI_j of the whole network by Step I.1 to Step II.3.

3.3.4 Fitness function

The fitness value is a measure of the goodness of a solution with respect to the original objective function. The choice of fitness function is also very critical because it must accurately measure the quality of the features described by the chromosome. The function should be computationally efficient since it is used many times to evaluate each and every solution. Consider an objective function *Obj* and a set of K constraints: $A1X \le B1$, $A2X \le B2,\ldots,A_KX \le B_K$. We first transform the constrained problem to an unconstrained one by setting the new objective function as:

$$Obj' = Obj + \beta \sum_{k=1}^{K} \max\left(A_k X - B_k, 0\right), \tag{14}$$

where β, a big enough positive number, is a penalty coefficient. Then the fitness function can be taken as the reciprocal of the new objective function multiple a constant.

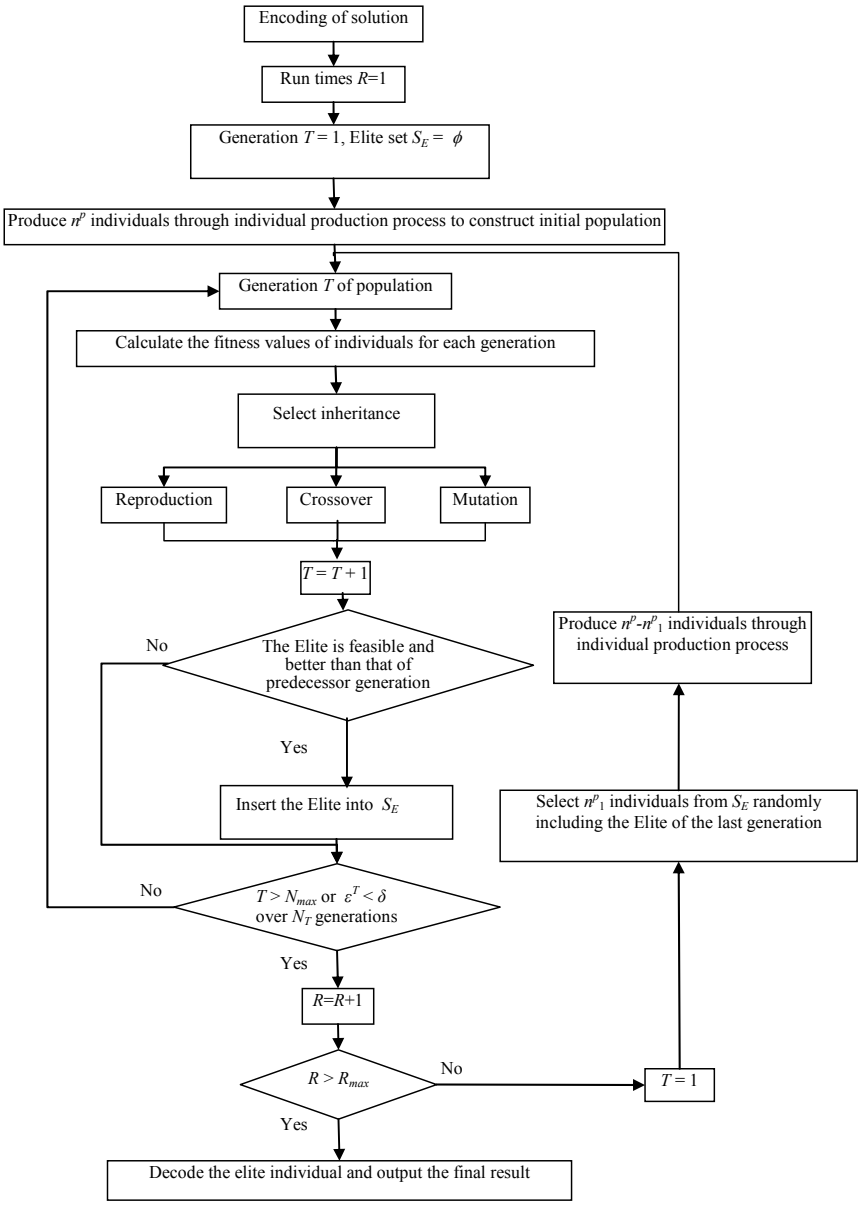
3.4 GA framework

The framework of our GA can be depicted by Figure 3. An initial population is composed of random feasible solutions generated by using the individual production process described in 3.2. The population size, n^p, is proportional to the size of supply chain network. Its specific value is often determined through experiment based on the balance between solution quality and consumed time.

Reproduction, crossover and mutation are executed in a parallel fashion. Reproduction guarantees individuals with the best fitness values in the current generation survive to the next generation as elite children. Crossover enables the algorithm to extract the best genes from different individuals and recombine them into potentially superior children. Mutation introduces diversity of a population and thereby increases the likelihood that the algorithm will generate individuals with better fitness values. The crossover operator is applied to each selected pair of parent chromosomes with probability p^{cross}, and the mutation operator is applied to each offspring with probability p^{mut}. Thus the next generation consists of $p^{cross} \cdot n^p$ crossover individuals, $p^{mut} \cdot n^p$ mutation

individuals, and elite individuals. It is much easier for GA to fall in pre-mature convergence if more same elite individuals are kept in the next generation, because they will dominate the whole process quickly. Thus the count of elite solutions is set be one in our implementation. N_{max} is the maximum number of iterations to perform. The algorithm also terminates if the weighted average change in the fitness function value ε^T over N_T generations is less than a parameter δ.

Figure 3 Sketch of the GA framework



In each generation, if the best individual (elite solution) is feasible and better than its parents, it will be added to a feasible individual set S_E. In order to get high quality solution, we execute the genetic evolution process R_{max} times. Since the second run, the initial population is composed of two parts of feasible individuals. The first part have $n^p{}_1$ individuals, which are randomly picked up from set S_E, including the best individual found so far. The second part have $n^p - n^p{}_1$ individuals, which will be produced through the individual production process.

In order for the algorithm to achieve high solution quality, we have implemented a multi-restart strategy. After an evolution iterates for N_{max} generations or $\varepsilon^T < \delta$, it restarts with an updated initial population, which is indicated by the loop on the right-hand-side. The elite solution of the last generation in the previous evolution is copied into the previous population, so that the updated population is an improved one. At the same time, the elite solution in each generation will be stored in a feasible solution pool, S_E, only if it is better than the previous generation. When an evolution ends, $n^p1–1$ individuals will be randomly selected from S_E to ensure better searching performance. In addition, the $n^p–n^p1$ initial individuals are reproduced through the operators to prevent premature convergence.

4 Computational experiment

In order to evaluate solution quality of our new GA algorithm, we solved the same 45 test instances with 20, 40 and 80 nodes as in Li and Jiang (2012). We implemented our GA algorithm in MATLAB using the GA Toolbox (Mathworks, 2008). We also solved the instances to optimality through the exact piecewise linear MIP method proposed by Magnanti (2006), the iterative LP heuristic (Shu and Karimi, 2009) and the CP-GA algorithm (Li and Jiang, 2012). Both the exact MIP and Iterative LP methods are implemented with the same stopping criteria as in Magnanti (2006) and Shu and Karimi (2009) using the version of CPLEX 12.1 (ILOG, 2010). All the computations were executed on a Pentium IV PC with 3.2 GHz CPU and 1G RAM. The number of evolutions parameter R_{max} is set to be 5, 10, 20 for the network size of 20, 40, 80 nodes, respectively. We also set the total number of N^{max} to be 300, 600, 1000, and the population size of 150, 200, 300 for instances with 20, 40, 80 nodes, respectively. The crossover probability p^{cross} and the mutation probability p^{mut} are set 0.3 and some 0.7 respectively (Li and Jiang, 2012).

Due to the probabilistic nature of GA, we perform our GA for each instance five times and record the worst and best solutions. The results are reported in Tables 1 to 3, where Gap (%) denotes the percentage of deviation of the obtained solution with respect to the optimal solution found by the exact MIP method. CPU denotes the CPU time (in seconds) for finding the best solution.

Table 1 Computational results for the 20-node problem instances

Instance #	*No. of arcs*	*MIP CPU*	*Iterative LP*		*CP-GA*		*New GA*	
			GAP (%)	*CPU*	*GAP (%)*	*CPU*	*GAP (%)*	*CPU*
1	32	0	5.83	0.02	[0.00, 0.00]	2.22	[0.00, 0.00]	9.52
2	35	16	2.20	0.05	[1.04, 1.04]	6.00	[0.83, 1.45]	12.20
3	36	1	1.41	0.02	[0.00, 0.00]	< 0.01	[0.00, 0.00]	2.42
4	30	1	0.00	0.06	[1.33, 1.46]	20.72	[1.33, 1.48]	23.11
5	27	4	3.31	0.11	[0.11, 0.11]	< 0.01	[0.11, 0.11]	1.75
6	50	0	0.21	0.11	[0.00, 0.03]	1.72	[0.00, 0.03]	3.63
7	30	2	0.00	0.03	[0.44, 0.94]	10.56	[0.56, 0.98]	17.25
8	39	1	3.01	0.06	[0.03, 0.03]	< 0.01	[0.03, 0.03]	2.03
9	24	0	0.03	0.02	[1.37, 3.31]	45.84	[1.20, 4.45]	63.22
10	33	1	10.59	0.08	[1.46, 1.46]	0.55	[1.46, 1.46]	1.92
11	28	0	1.16	0.14	[0.00, 0.11]	4.80	[0.00, 0.11]	8.55
12	35	0	2.24	0.02	[3.86, 3.92]	3.86	[1.66, 3.40]	7.56
13	44	0	3.17	0.11	[0.00, 0.82]	3.44	[0.00, 0.95]	9.12
14	43	0	3.53	0.03	[0.00, 0.00]	< 0.01	[0.00, 0.00]	2.02
15	40	0	8.40	0.08	[0.01, 0.01]	2.02	[0.00, 0.45]	7.74
Average	35	1.73	2.99	0.06	[0.64, 0.88]	6.87	[0.48, 0.99]	11.47

Table 2 Computational results for the 40-node problem instances

Instance #	*No. of arcs*	*MIP CPU*	*Iterative LP*		*CP-GA*		*New GA*	
			GAP (%)	*CPU*	*GAP (%)*	*CPU*	*GAP (%)*	*CPU*
1	75	178	1.48	0.16	[0.31, 0.70]	23.89	[0.16, 0.83]	43.52
2	84	196	2.24	0.2	[0.11, 0.34]	44.42	[0.12, 0.33]	72.40
3	89	2	6.84	0.13	[0.35, 0.43]	7.91	[0.17, 0.41]	19.85
4	70	247	1.01	0.11	[0.70, 1.40]	32.78	[0.44, 1.42]	78.43
5	85	8	2.46	0.09	[1.60, 2.93]	38.17	[1.17, 2.76]	82.65
6	70	52	1.42	0.14	[0.39, 1.43]	37.70	[0.41, 1.45]	52.30
7	77	81	12.99	0.17	[0.12, 1.75]	47.25	[0.14, 1.70]	71.21
8	49	141	1.94	0.22	[3.20, 3.51]	80.14	[1.51, 3.46]	124.35
9	58	2,023	4.17	0.08	[1.36, 3.68]	51.89	[0.85, 2.90]	93.10
10	64	77	0.73	0.06	[1.10, 2.01]	46.97	[1.06, 1.86]	82.32
11	89	32	2.99	0.11	[0.20, 0.94]	45.98	[0.24, 0.79]	67.54
12	75	26	0.67	0.13	[0.66, 0.88]	27.77	[0.66, 0.87]	52.37
13	69	1,9313	3.99	0.2	[0.35, 1.74]	54.64	[0.22, 1.56]	95.61
14	85	392	3.41	0.06	[1.85, 1.95]	32.25	[0.91, 1.85]	65.22
15	76	4,448	0.46	0.14	[0.40, 1.30]	35.00	[0.33, 1.75]	74.35
Average	74	1,814.40	3.12	0.13	[0.85, 1.67]	40.45	[0.56, 1.60]	71.68

Table 3 Computational results for the 80-node problem instances

Instance #	*No. of arcs*	*MIP CPU*	*Iterative LP*		*CP-GA*		*New GA*	
			GAP (%)	*CPU*	*GAP (%)*	*CPU*	*GAP (%)*	*CPU*
1	149	29,117	3.46	0.33	[1.86, 3.09]	170.44	[1.25, 3.14]	280.92
2	134	11,209	1.60	0.27	[1.13, 2.91]	169.94	[0.83, 2.43]	261.30
3	213	18,031	3.33	0.27	[2.74, 4.91]	186.20	[1.45, 5.23]	297.65
4	111	36,000	2.33	0.30	[3.14, 5.01]	136.81	[1.34, 4.19]	231.14
5	142	18,409	4.39	0.42	[5.74, 6.53]	146.50	[2.11, 4.38]	254.23
6	93	36,000	5.75	0.34	[7.78, 9.08]	153.63	[3.06, 6.92]	263.11
7	102	61,232	7.70	0.45	[3.11, 4.40]	160.59	[1.22, 4.05]	257.66
8	139	47,501	4.92	0.19	[4.63, 6.88]	169.66	[2.25, 6.13]	278.32
9	172	27,745	1.65	0.36	[1.70, 2.59]	63.31	[0.93, 2.61]	121.50
10	116	33,213	4.63	0.36	[3.91, 4.95]	166.11	[2.36, 4.62]	255.72
11	141	29,098	1.72	0.34	[5.39, 7.30]	171.70	[2.76, 5.31]	285.45
12	87	36,000	1.95	0.75	[9.03, 11.00]	154.14	[4.86, 8.40]	266.29
13	171	7,256	1.67	0.41	[1.86, 2.65]	157.77	[0.88, 2.95]	264.35
14	108	18,174	2.51	0.31	[3.90, 8.71]	164.06	[1.90, 7.19]	258.74
15	112	25,215	4.87	0.27	[4.87, 7.52]	152.55	[3.25, 5.45]	254.50
Average	133	28,946.67	3.50	0.36	[4.05, 5.84]	154.89	[2.03, 4.87]	255.39

For the 15 20-node instances, our new GA is able to reduce the average gap of CP-GA for the optimal solutions from 0.64% to 0.48% with slightly more computational time. It also outperforms the Iterative LP method with a gap of 2.99%.

For the 15 40-node instances, the GA further improves CP-GA solution quality form 0.85% to 0.56%. The average time for the new GA to obtain the best almost doubles the time of CP-GA, but much shorter than the exact MIP method which takes about 30 minutes on average to reach optimal solutions.

For the 15 80-node instances, the GA achieves the largest improvement over the CP-GA by reducing the optimality gap from 4.05% to 2.03%. Although it spends 255 seconds on average for finding the best solutions, it is significantly more efficiently than the exact MIP method which spends about eight hours on average to reach optimality.

To examine the performance of our GA on solving larger instances, we adapted the 38 benchmark instances in Humair and Willems (2011) with size up to 2,025 nodes and 16,225 arcs. The computational results and comparison with the algorithms of Humair and Willems (2011) are shown in Table 4. The efficiency of our GA algorithm is evident in that with the same algorithmic setting, 52.6% of the instances take less than one minute, 26.3% of the instances take about ten minutes. The longest time needed is less than two hours (for the No. 38 instances). The best objective value of each instance is not directly comparable. However, note that for instances nos. 1, 6, 14 and 38, our GA finds solutions with lower objective values (even with capacity constraint).

Table 4 The results and comparison with Humair and Willems (2011) on the 38 benchmark instances

Chain	*Stages*	*Links*	*Max chain length*	*GNA algorithm*		*HGNA Heuristic*		*TGNA Heuristic*		*New GA*	
				Time (secs)	*Best obj cost*	*Time (secs)*	*Best obj cost*	*Time (secs)*	*Best obj cost*	*Time (secs)*	*Best obj cost*
1	8	10	38	0	3.35E+04	0	3.35E+04	0.015	3.35E+04	2.3124	2.05E+04
2	13	13	64	0.03	9.51E+06	0.03	9.51E+06	0.032	9.51E+06	3.9143	2.94E+07
3	17	18	79.8	0.11	6.94E+06	0.06	6.94E+06	0.062	6.94E+06	3.0603	1.40E+07
4	22	39	204	0.63	4.90E+04	0.53	4.90E+04	0.64	4.90E+04	3.6898	1.41E+05
5	27	31	47.35	0	2.01E+06	0	2.01E+06	0.015	2.01E+06	3.2587	3.86E+06
6	28	28	96	0	7.78E+04	0.02	7.78E+04	0.016	7.78E+04	14.7603	1.30E+03
7	38	78	85	1.47	5.22E+06	0.28	5.22E+06	1.453	5.22E+06	9.9138	1.08E+07
8	40	48	91.04	0.28	1.63E+06	0.27	1.63E+06	0.265	1.63E+06	2.989	3.90E+06
9	49	52	47.38	0.02	1.12E+06	0.02	1.12E+06	0	1.12E+06	6.9549	2.56E+06
10	58	176	162	0.08	1.21E+06	0.06	1.21E+06	0.078	1.21E+06	6.2969	2.38E+06
11	68	108	60	9.5	1.12E+07	0.66	1.12E+07	2.547	1.12E+07	13.4718	1.97E+07
12	88	107	108.6	10.67	7.35E+06	0.98	7.35E+06	2.187	7.35E+06	18.643	1.92E+07
13	108	452	26	22.59	6.09E+06	0.02	6.09E+06	0.094	8.34E+06	7.3165	1.75E+07
14	116	119	128.32	0.13	7.16E+04	0.14	7.16E+04	0.14	7.16E+04	23.3575	5.36E+04
15	133	164	26	0.02	1.16E+06	0.02	1.16E+06	0.015	1.16E+06	34.7592	2.87E+06
16	145	224	163	30.03	4.64E+06	24.92	4.64E+06	26.375	4.64E+06	43.8395	9.59E+06
17	152	211	57	0.02	1.09E+06	0.02	1.09E+06	0.031	1.09E+06	31.9196	3.92E+06
18	154	224	100	2.09	9.75E+04	2.47	9.75E+04	1.547	9.75E+04	43.1763	3.18E+05
19	156	263	125	3,103.63	3.15E+05	13.84	3.18E+05	5.031	3.19E+05	68.4257	9.13E+05

Table 4 The results and comparison with Humair and Willems (2011) on the 38 benchmark instances (continued)

Chain	*Stages*	*Links*	*Max chain length*	*GNA algorithm*		*HGNA Heuristic*		*TGNA Heuristic*		*New GA*	
				Time (secs)	*Best obj cost*	*Time (secs)*	*Best obj cost*	*Time (secs)*	*Best obj cost*	*Time (secs)*	*Best obj cost*
20	156	169	160.9	804.39	2.91E+05	2.3	3.02E+05	4.797	3.01E+05	32.5219	5.65E+05
21	186	359	96	140.34	1.08E+06	6.61	1.08E+06	7	1.08E+06	56.5735	2.51E+06
22	253	253	691	12.19	1.94E+06	11.69	1.94E+06	12.218	1.94E+06	132.0187	2.42E+06
23	271	524	77	176,722.45	4.26E+05	56.53	4.26E+05	1.844	4.26E+05	156.3966	1.29E+06
24	334	1,245	68.53	10,435.67	1.41E+07	35.47	1.44E+07	8.453	1.45E+07	275.1461	3.49E+07
25	409	853	82	883,312.55	1.14E+06	143.52	1.14E+06	9.016	1.15E+06	323.7947	3.38E+06
26	468	605	394.07	8.38	4.53E+06	3.58	4.53E+06	8.375	4.53E+06	519.8254	8.85E+06
27	482	941	105	705,636.17	8.32E+05	221.83	7.53E+05	7.375	8.51E+05	245.1051	2.18E+06
28	577	2,262	123	103.02	4.63E+06	18.03	4.63E+06	26.656	4.73E+06	553.2243	1.98E+07
29	617	753	43	0.75	3.46E+05	0.75	3.46E+05	0.766	3.46E+05	540.8729	1.00E+06
30	626	632	71.05	0.69	9.60E+05	1.28	9.60E+05	0.688	9.60E+05	896.7473	2.48E+06
31	706	908	17.92	116.22	3.40E+07	5.99	3.41E+07	2.828	3.49E+07	367.3201	7.14E+07
32	844	1,685	112.2	3,089,722.84	7.09E+05	1,708.02	7.09E+05	31.093	7.09E+05	2024.9751	1.52E+06
33	976	1,009	72.36	0.28	4.22E+05	0.3	4.22E+05	0.281	4.22E+05	2205.3684	1.13E+06
34	1,206	4,063	89	12.66	8.64E+05	9.42	8.64E+05	8.594	8.64E+05	2524.3132	2.55E+06
35	1,386	1,857	81	37.88	1.79E+06	7.11	1.80E+06	16.203	1.81E+06	2827.3099	5.50E+06
36	1,451	4,812	49.55	126,583.03	1.24E+06	1,028.47	1.43E+06	13.765	1.43E+06	3467.2098	1.85E+06
37	1,479	2,069	27.85	17.28	9.99E+05	2.63	9.99E+05	6.875	9.99E+05	3549.1974	4.49E+05
38	2,025	16,225	26.03	577,190.78	1.10E+06	39.31	1.29E+06	16.938	1.53E+06	6128.8435	1.22E+06

5 Conclusions and future research

In this paper, we present a new GA algorithm to solve the inventory positioning problem with acyclic capacitated supply chain networks. We have implemented a direct integer encoding scheme to limit the size of solution space. Custom procedures and operators tailored for our problem are developed to randomly generate feasible solutions. We also designed and implemented multi-start strategy to improve the algorithm performance and prevent pre-mature convergence. We tested performance of our GA on the benchmark instances. Comparing with the hybrid CP-GA algorithm and the iterative LP heuristic, our GA appears to have better solution quality with reasonable computational time.

Several future research directions are promising. Our GA can be further improved by designing more sophisticated GA operators and fine tuning the crossover and mutation rates. With proper adjustment and modification, our GA approach framework can be adapted for cyclic networks, i.e. networks with loops often encountered in reverse logistics. Another interesting future research is to address the problem with explicit uncertainty about lead times.

Acknowledgements

This research was supported by Natural Science Foundation Project of CQ CSTC (2011BB0045). We also thank three anonymous referees for their valuable and constructive comments that help improve the contents and presentation of this paper.

References

Chung, J-W., Chung, S-M. and Oh, I-C. (2009) 'A hybrid genetic algorithm for train sequencing in the Korean railway', *Omega*, Vol. 37, No. 3, pp.555–565.

Davis, L. (1991) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Gen, M. and Cheng, R. (2000) *Genetic Algorithms and Engineering Optimizations*, Wiley, New York.

Gen, M., Choi, J. and Tsujimura, Y. (1999) 'Genetic algorithm for the capacitated plant location problem with single source constraints', *Proceedings of seventh European congress on intelligent techniques and soft computing, Session CD-7*.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Goldberg, D.E. and Lingle, R. (1985) 'Alleles, loci, and the traveling salesman problem', *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pp.154–159, Lawrence Erlbaum Associates, Hillsdale, NJ.

Grahl, J., Minner, S. and Dittmar, D. (2014) 'Meta-heuristics for placing strategic safety stock in multi-echelon inventory with differentiated service times', *Annals of Operations Research*, 24 July, pp.1–16, doi: 10.1007/s10479-014-1635-1.

Graves, S.C. and Willems, S.P. (2000) 'Optimizing strategic safety stock placement in supply chains', *Manufacturing and Service Operations Management*, Vol. 2, No. 1, pp.68–83.

Hadj-Alouane, A.B. and Bean, J.C. (1997) 'A genetic algorithm for the multiple-choice integer program', *Operations Research*, Vol. 45, No. 1, pp.92–101.

Harper, P.R., de Senna, V., Vieira, I.T. and Shahani, A.K. (2005) 'A genetic algorithm for the project assignment problem', *Computers and Operations Research*, Vol. 32, No. 5, pp.1255–1265.

Hua, Z. and Huang, F. (2006) 'A variable-grouping based genetic algorithm for large-scale integer programming', *Information Sciences*, Vol. 176, No. 19, pp.2869–2885.

Humair, S. and Willems, S.P. (2011) 'Optimizing strategic safety stock placement in general acyclic networks', *Operations Research*, Vol. 59, No. 3, pp.781–787.

Hung, P-C. and Chen, Y-P. (2006) 'iECGA: integer extended compact genetic algorithm', *GECCO'06*, Seattle, Washington, USA, 8–12 July 2006, pp.1415–1416.

ILOG (2010) *ILOG Cplex 12.1 User's Manual*, IBM, Mountain View, CA.

Jaramillo, J.H., Bhadury, J. and Batta, R. (2002) 'On the use of genetic algorithms to solve location problems', *Computers and Operations Research*, Vol. 29, No. 6, pp.761–779.

Li, H. and Jiang, D. (2012) 'New model and heuristics for safety stock placement in general acyclic supply chain networks', *Computers and Operations Research*, Vol. 39, No. 7, pp.1333–1344.

Magnanti, T.L., Shen, Z-J., Shu, J., Simchi-Levi, D. and Teo, C-P. (2006) 'Inventory placement in acyclic supply chain networks', *Operations Research Letters*, Vol. 34, No. 2, pp.228–238.

Mathworks (2008) *User's Guide for Genetic Algorithm and Direct Search Toolbox*, Mathworks Inc., Natik, MA [online] http://www.mathworks.com/help/releases/R13sp2/pdf_doc/gads/gads_tb.pdf (accessed January 2014).

Minner, S. (1997) 'Dynamic programming algorithms for multi-stage safety stock optimization', *OR Spectrum*, Vol. 19, No. 4, pp.261–271.

Minner, S. (2000) *Strategic Safety Stocks in Supply Chains*, Springer, Berlin.

Moon, I., Lee, S. and Bae, H. (2008) 'Genetic algorithms for job shop scheduling problems with alternative routings', *International Journal of Production Research*, Vol. 46, No. 10, pp.2695–2705.

Orero, S.O. and Irving, M.R. (1997) 'A combination of the genetic algorithm and Lagrangian relaxation decomposition techniques for the generation unit commitment problem', *Electric Power Systems Research*, Vol. 43, No. 3, pp.149–156.

Palmer, C.C. and Kershenbaum, A. (1995) 'An approach to a problem in network design using genetic algorithms', *Network*, Vol. 26, No. 26, pp.151–163.

Pruettha, N. and Konlakarn, M. (2001) 'An adaptive penalty function in genetic algorithms for structural design optimization', *Computers and Structures*, Vol. 79, No. 29, pp.2527–2539.

Sahni, S. (1974) 'Computationally related problems', *SIAM Journal on Computing*, Vol. 3, No. 4, pp.262–279.

Shu, J. and Karimi, I.A. (2009) 'Efficient heuristics for inventory placement in acyclic networks', *Computers and Operations Research*, Vol. 36, No. 11, pp.2899–2904.

Willems, S.P. (2008) 'Real-world multi-echelon supply chains used for inventory optimization', *Manufacturing and Service Operations Management*, Vol. 10, No. 1, pp.19–23.

Yokota, T., Gen, M. and Li, Y.X. (1996) 'Genetic algorithm for non-linear mixed integer programming problems and its applications', *Computers and Industrial Engineering*, Vol. 30, No. 4, pp.905–917.

Zhou, G. and Gen, M. (1999) 'Genetic algorithm approach on multicriteria minimum spanning tree problem', *European Journal of Operational Research*, Vol. 114, No. 1, pp.141–152.

Zhou, G., Min, H. and Gen, M. (2003) 'A genetic algorithm approach to the bi-criteria allocation of customers to warehouses', *International Journal of Production Economics*, Vol. 86, No. 1, pp.35–45.