A Sequence-Based Metaheuristics for Tactical Distribution Problems in Closed-Loop Supply Chains

Pierre Desport, David Lesaint, Frédéric Lardeux, Carla Di Cairano - Gilfedder, Anne Liret, Gilbert Owusu

A Sequence-Based Metaheuristic For Tactical Distribution Planning In Closed-Loop Supply Chains

Pierre Desport [*,**,***] **Frédéric Lardeux** [*] **David Lesaint** [*]
Carla Di Cairano-Gilfedder [***] **Anne Liret** [***]
Gilbert Owusu [***]

[*] *LERIA, EA 2645, UNIV Angers, 2 Bd Lavoisier, 49045 Angers, France (e-mail: firstname.lastname@univ-angers.fr)*
[**] *LIFAT EA 6300, ROOT ERL CNRS 7002, Université de Tours, 64 avenue Jean Portalis, Tours 37200, France (e-mail: firstname.lastname@univ-tours.fr)*
[***] *British Telecom, Adastral Park, Martlesham Heath, UK (e-mail: firstname.lastname@bt.com)*

Abstract: We present a metaheuristic for planning the distribution of items in closed-loop supply chains. This metaheuristic composes sequences of transfer and repair actions to generate plans iteratively. It uses a local search algorithm based on an efficient data structure to construct and select improving sequences at each step. An experimental comparison with a mixed integer programming approach shows its scalability and robustness on a variety of instances. We also study and discuss the ability to support different distribution policies.

Keywords: Supply Chain Planning, Metaheuristics, Distribution, Transport

1 Introduction

Many organisations in equipment-intensive sectors (automotive, electronics, telecommunications, ...) implement closed-loop supply chains to recapture economic value from used or defective parts. Closed-loop supply chains (CLSC) extend the "forward flow" model of traditional supply chains by integrating reverse logistics operations including repair, recycling or disposal (**????**). In field service organisations for instance, CLSC typically combine two flows: the replenishment of field depots with spare parts, and the transfer of used parts back to warehouse centres before repair and possible reuse. Figure 1 illustrates a three-echelon CLSC in this context where a warehouse centre relies on a repair site to reinject repaired parts in the supply flow.

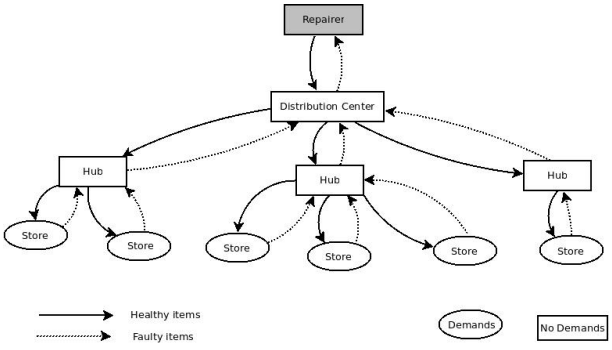


Fig. 1. A 3-level closed-loop supply chain

CLSC inventory systems control all repair and transfer activities (supply of spare parts and return of used parts) across the different sites of the supply chain (warehouse centres, repair centres, hubs, field depots ...). In some environments (e.g. field maintenance operations), the demand for spare parts for each site and each time period is known beforehand with certainty. In this case, a centralised proactive approach may be applied to plan these activities ahead of time and minimise costs relating to backordering, storage, transport and repair. **?** propose a mixed integer program (MIP) to solve this tactical distribution planning problem (TDPP). The model uses a weighted objective function and combines conservation-of-flow constraints with a priority rule that forbids backordering on a site in the presence of spares. This rule makes the problem NP-hard and the authors introduce a metaheuristic called BIS that compares favourably against a MIP method over pseudo-random instances.

BIS is a best-improvement local search algorithm using diversification. It builds plans iteratively by selecting and adding at each step a sequence of unitary actions (supply, return and/or repair of a single part). In this paper, we provide a detailed description of the two components of BIS, namely, the local search algorithm and the sequence construction and selection procedure which uses a dedicated data structure to efficiently update the set of feasible sequences and resulting gains. The rest of the paper is organised as follows. Section 2 gives a brief overview of the MIP model for TDPP. Section 3 introduces BIS and section 4 presents experimental results comparing BIS and MIP. Section 5 then discusses how different weightings

of the objective function affect the distribution of items. Section 6 concludes.

2 Tactical Distribution Planning Problem

A TDPP instance is primarily defined by a CLSC network and a discrete planning horizon. As shown in Figure 1, a CLSC network consists of supply, return and repair routes connecting the different sites. Each route comes with a fixed schedule that defines the possible start and end times for actions planned along this route (e.g., 1-day long weekly shipments starting on Mondays). Each site initially holds a predefined number of spare parts (aka. healthy items) and used parts (aka. faulty items). The "demand" forecast then defines the number of healthy items that should be taken out of inventory on each site and at each point in time (stock permitting) and the number of faulty items that will be dropped on the site and enter the supply chain. These quantities are considered identical in some environments (e.g., field repair services) which is the assumption made here.

The objective is to determine an optimal and feasible plan of actions. Each action is defined by the choice of a route, a start time, and a quantity representing the number of healthy or faulty items acted upon (transfers or repairs). All actions must comply with the above mentioned topological and calendar constraints, and the conservation-of-flow constraints enforced on each site over healthy and faulty items. To comply with standard practices, an additional rule applies that forbids backordering if spares are locally and readily available on a site, i.e., satisfying local demand takes priority. Lastly, the objective is cast as a weighted sum of linear cost functions on the total number of items that are backordered, held in inventory, transferred or repaired. No start-up penalty applies and the weights are used-defined to allow different prioritisation strategies.

Table 1. Constants and variables of a TDPP instance

Constants	
demand forecast	$d : L \times T \to \mathbb{N}$
storage unit costs	$\mu^{s^h}, \mu^{s^f} : L \to \mathbb{R}$
backordering unit costs	$\mu^b : L \to \mathbb{R}$
transfer unit costs	$\mu^{m^h}, \mu^{m^f} : L \times L \to \mathbb{R}$
repair unit costs	$\mu^{m^r} : L \to \mathbb{R}$
weights	$\omega^s, \omega^b, \omega^{m^{hf}}, \omega^{m^r} \in \mathbb{N}$
Variables	
storage	$s^h, s^f : L \times T \to \mathbb{N}$
backorders	$b : L \times T \to \mathbb{N}$
transfers	$m^h, m^f : L \times L \times T \times T \to \mathbb{N}$
repairs	$m^r : L \times T \times T \to \mathbb{N}$
costs	$c, c^s, c^b, c^{m^{hf}}, c^{m^r} \in \mathbb{R}$

? propose a MIP to model this problem which we briefly introduce. L and T denote, respectively, the set of sites and the planning horizon viewed as an ordered set of time buckets. Table 1 introduces the constants and variables of the model. Constants include the demand forecast d, the unit costs μ^* and the user-defined weights w^*. Variables model the actions of the plan (in bold style) and the resulting state of the supply chain over the temporal horizon. For two sites l and l', and time buckets t and t', variable $m^h_{l,l',t,t'}$ (respectively, $m^f_{l,l',t,t'}$) denotes the number of healthy (resp., faulty) items transferred from l at t and reaching l' at t'. Likewise, variable $m^r_{l,t,t'}$ denotes the number of faulty items whose repair starts at t and ends at t' on site l. The constraints (not shown here) link state and action variables to enforce the rules relating to routing, scheduling, flow preservation, and flow prioritisation. In particular, any action variable corresponding to a non-existant route is set to 0. Note also that any assignment of the action variables completely determine the state variables. The equations below show the objective function c to minimise and its different components.

$$c^s = \sum_{l \in L, t \in T} (s^h_{l,t} \times \mu^{s^h}_l + s^f_{l,t} \times \mu^{s^f}_l)$$

$$c^b = \sum_{l \in L, t \in T} b_{l,t} \times \mu^b_l$$

$$c^{m^{hf}} = \sum_{\substack{l,l' \in L,\\ t,t' \in T}} (m^h_{l,l',t,t'} \times \mu^{m^h}_{l,l'} + m^f_{l,l',t,t'} \times \mu^{m^f}_{l,l'})$$

$$c^{m^r} = \sum_{\substack{l \in L,\\ t,t' \in T}} (m^r_{l,t,t'} \times \mu^{m^r}_l)$$

$$c = \omega^s \times c^s + \omega^b \times c^b + \omega^{m^{hf}} \times c^{m^{hf}} + \omega^{m^r} \times c^{m^r}$$

3 Metaheuristic

This section presents the metaheuristic BIS (Best Improving Sequence) for solving TDPP. BIS does not operate on the solution representation of the MIP model. Rather, it constructs a plan by repeatedly adding sequences of unitary actions (i.e., actions that only involve a single item). As explained below, sequences are constructed over the space-time graph of the problem based on the current plan, filtered and scored using a dedicated data structure, and selected using a probabilistic diversification policy.

3.1 Solution representation and neighbourhood

The space-time graph of a TDPP instance, denoted G^*, represents all the activity routes allowed at each point in time. Its nodes correspond to the pairings of sites and time buckets ($L \times T$) and its arcs link the nodes consistently with the network topology, schedule and lead time constraints. G^* is the union of 3 subgraphs G^h, G^f and G^r that represent the supply links, the return links and the repair links, respectively. Any planned action on n items (i.e., the assignment of value n to an action variable of the MIP that maps to a route) may be viewed as a labelling of an arc of G^* with quantity n. Therefore, any plan (i.e., any set of actions) determines itself a particular labelling of G^* with quantities.

Since every action on n items splits into n unitary actions, every plan is itself defined by a multiset of unitary actions. A plan may also be defined as a set of sequences by clustering these actions into sequences consistently with

G^* (i.e., into chains of unit actions over G^*). BIS adopts this view to construct a plan by assembling sequences iteratively. At each iteration, it searches for a sequence amongst the set of feasible sequences and adds it to the current plan. This addition operation boils down to a set union operation between the multiset of actions defining the plan and the set of actions defining the sequence. Formally, the neighbourhood N of a plan P is the set $N(P) = \{P \cup S | S \in SEQ\}$ where SEQ denotes the set of sequences consistent with P.

Algorithm 1 describes the global structure of BIS. BIS relies on Algorithm 2 for constructing and selecting a sequence at each step which we present in Section 3.2. By design, BIS cannot remove sequences to "undo" a plan. Every choice of sequence is therefore critical in the construction process and we choose to adopt a best improvement sequence selection approach. To avoid local optima, we introduce stochasticity in Algorithm 2 and we restart the search for a plan a predefined number of times. This process is inspired from Variable Neighbourhood Search (VNS) (**?**) which increases the size of the neighbourhood at each restart of the algorithm. It is also inspired by Simulated Annealing which, contrary to our approach, evolves from a large to a small neighbourhood (**?**). Another common issue with best improvement selection is the computational cost incurred by the evaluation of all neighbours. We make use of a specific data structure to this effect which is presented in Sections 3.3 and 3.4.

Algorithm 1 BIS

Input: an initial plan P_0, a number of restarts n_{max}, an initial probability α_0, a control parameter v
Output: a plan

$P^* \leftarrow P_0$
$\alpha \leftarrow \alpha_0$
$n \leftarrow 1$
while $n \leq n_{max}$ **do**
  $P \leftarrow P_0$;
  Build sequence seq using Algorithm 2 with parameters P and α;
  while $seq \neq seq_\emptyset$ **do** ▷ $seq_\emptyset$ is the empty sequence
    $P \leftarrow P \cup seq$
    Build sequence seq using Algorithm 2 with parameters P and α;
  end while
  if $c(P) \leq c(P^*)$ **then** ▷ c is the cost function
    $P^* \leftarrow P$
  end if
  $\alpha \leftarrow 1 - ((1-v)^n \times (1-\alpha_0))$
  $n \leftarrow n + 1$
end while
return P^*

3.2 Sequence selection

Every sequence of SEQ is a tuple of unitary actions that corresponds to a path in G^*. We shall denote by $X_{l,l',t,t'}$ a unit action of type X that starts at t from site l and ends at t' on site l'. Specifically, $B_{l,l',t,t'} \in G^f$ (respectively, $R_{l,l,t,t'} \in G^r$, $A_{l,l',t,t'} \in G^h$) will denote a return (resp., repair, supply) action. We shall also use subscript symbol $*$ to indicate any particular location or time bucket. For instance, $A_{l,*,t,*}$ will denote any supply action starting from site l at t. We list below the chaining rules that prevail when constructing a sequence of SEQ:

(1) An action must start from the end site of the previous action, if any.
(2) An action must not start before the end time bucket of the previous action, if any.
(3) An item must be available on the site of an action when it starts.
(4) The on-hand inventory on the site of the action initiating the sequence must be sufficient not to cancel subsequent unitary actions that are already planned.

Algorithm 2 describes the selection and construction of a sequence of unit actions. It is based on two functions, denoted $gainC$ and $gainI$, which compute the improvement that a unit action could bring to a plan (see Section 3.3) and a variable corresponding to the best possible action following a given action ($next(y)$ is the best action following action y). Algorithm 2 uses probability α to introduce stochasticity when selecting an improving sequence. Every time we select a unit action to extend the sequence, we might select another improving action than the best one or end the sequence with a probability equals to α. Probability α evolves at each restart in BIS following an arithmetico-geometric series. Given α_0 the initial probability and v a control parameter, probability α_n at the n-th restart is defined as follows:

$$\alpha_n = 1 - ((1-v)^n \times (1-\alpha_0))$$

Parameter v ranges in $[0, 1]$ and is used to control the range of values for α. Based on experiments, we deduced that α_0 should be set between 0 and 0.15 to ensure a good balance between intensification and diversification. Depending on the number of restarts, v is adjusted to make α range from 0 to 0.15.

Algorithm 2 Construction of a sequence

Input: a plan P, a probability α
Output: a sequence seq

$seq \leftarrow seq_\emptyset$
if $max(gainI(X_{*,*,*,*})) > 0$ **then**
  $p \leftarrow Random(0, 1)$ ▷ a random value in [0,1]
  if $p \geq \alpha$ **then**
    Select y such that $gainI(y) = max(gainI(X_{*,*,*,*}))$
  else
    Select y such that $gainI(y) > 0$
  end if
  repeat
    Append y to seq
    $p \leftarrow Random(0, 1)$
    if $p \geq \alpha$ **then**
      $y' \leftarrow next(y)$ ▷ $next(y)$ returns NULL if there is no improving action following y.
    else
      Select y' such that $gainC(y') > 0$ and y' follows y
    end if
    $y \leftarrow y'$
  until $y = NULL$
end if
return seq;

3.3 Gain functions

We present here the definition of functions $gainI$ and $gainC$ and the set of variables they rely upon.
Boolean variable $a^h_{l,t}$ (resp. $a^f_{l,t}$) denotes the possibility to

send an healthy (resp. faulty) item from site l at time bucket t. This action is possible only if items are available at t and if it does not cancel another action planned later on. Note that in case of healthy items, sending an item might cause a backorder on a subsequent time bucket but must not cause a backorder on the current time bucket to remain consistent with the TDPP constraints.

$$\forall l \in L, t \in T, a^h_{l,t} = \begin{cases} 1 \text{ if } s^h_{l,t} > 0 \text{ and } \forall t' \in]t, Tmax], \\ \sum_{\substack{l' \in L, \\ t'' > t'}} m^h_{l,l',t',t''} > 0 \rightarrow s^h_{l,t'} \geq 1 \\ 0 \text{ otherwise} \end{cases}$$

$$\forall l \in L, t \in T, a^f_{l,t} = \begin{cases} 1 \text{ if } s^f_{l,t} > 0 \text{ and } \forall t' \in]t, Tmax], \\ \sum_{\substack{l' \in L, \\ t'' > t'}} m^f_{l,l',t',t''} > 0 \rightarrow s^f_{l,t'} \geq 1 \\ 0 \text{ otherwise} \end{cases}$$

We also define variables that represent the number of backorders generated or avoided by applying a supply action and its impact on storage for both the source and target sites of the action.

$\forall l \in L, t \in T, t' > t,$
//number of backorders avoided - target site
$\mathbf{ddStr}(l,t,t') = |\{t'' | t' > t'' \geq t \text{ and } b_{l,t''} > 0\}|$
//number of backorders generated - source site
$\mathbf{dd}(l,t,t') = |\{t'' | t' > t'' \geq t \text{ and } s^h_{l,t''} = 0\}|$
//impact on storage - target site
$\mathbf{dhs}(l,t,t') = |\{t'' | t' > t'' \geq t \text{ and } b_{l,t''} = 0\}|$
//impact on storage - source site
$\mathbf{dhsStr}(l,t,t') = |\{t'' | t' > t'' \geq t \text{ and } s^h_{l,t''} > 0\}|$

Based on these variables, functions $gainC$ and $gainI$ compute the maximum gain a unitary action can bring to the current plan. $gainC(x)$ computes the improvement that unitary action x can bring to a sequence. In other words, when building a sequence, $gainC$ will help to decide which action should be chosen to continue the sequence, if any. This function is computed recursively and takes into account the improvement that could be brought by an action following the one we are evaluating.

$\mathbf{gainC}(A_{l,l',t,t'}) = max(\ ddStr(l',t',Tmax+1) * \mu^b_{l'} * \omega^b$
$-\mu^{m^h}_{l,l'} * \omega^{m^{hf}} - dhs(l',t',Tmax+1) * \mu^{s^h}_{l'} * \omega^s\ ,$
$max(\{gainC(A_{l',*,t'',*}) - \mu^{m^h}_{l,l'} * \omega^{m^{hf}} - dhs(l',t',t'') * \mu^{s^h}_{l'} * \omega^s$
$+ddStr(l',t',t'') * \mu^b_{l'} * \omega^b | t'' \geq t', b_{l',t''} = 0\}))$
$\mathbf{gainC}(R_{l,l,t,t'}) = max(\ ddStr(l,t',Tmax+1) * \mu^b_l * \omega^b - \mu^{m^r}_l * \omega^{m^r}$
$-dhs(l,t',Tmax+1) * \mu^{s^h}_l * \omega^s\ ,$
$max(\{gainC(A_{l,*,t'',*}) - \mu^{m^r}_l * \omega^{m^r} - dhs(l,t',t'') * \mu^{s^h}_l * \omega^s$
$+ddStr(l,t',t'') * \mu^b_l * \omega^b | t'' \geq t', b_{l,t''} = 0\}))$
$\mathbf{gainC}(B_{l,l',t,t'}) = max(\ -\mu^{m^f}_{l,l'} * \omega^{m^{hf}} - (Tmax - t') * \mu^{s^f}_{l'} * \omega^s\ ,$
$max(\{gainC(B_{l',*,t'',*}) - \mu^{m^f}_{l,l'} * \omega^{m^{hf}} - (t'' - t') * \mu^{s^f}_{l'} * \omega^s | t'' \geq t'\})\ ,$
$max(\{gainC(R_{l',l',t'',*}) - \mu^{m^f}_{l,l'} * \omega^{m^{hf}} - (t'' - t') * \mu^{s^f}_{l'} * \omega^s | t'' \geq t'\}))$

Note that to avoid multiple computation of the same action, we initialize variable $next(x)$ when computing $gainC$. This variable returns the best action that could follow x or $NULL$ if no action could improve the sequence after applying x.

$gainI(x)$ defines the maximum gain that a sequence initialised with unitary action x could bring to the current solution. This function differs from $gainC$ as $gainC$ only refers to actions included in a sequence and not to actions initialising a sequence. Note that $gainI$ assigns a negative value to actions which are inconsistent with the current plan so that they will never be selected to build a sequence.

$$gainI(A_{l,l',t,t'}) = \begin{cases} -1 \textbf{ if } a^h_{l,t} = 0 \\ gainC(A_{l,l',t,t'}) + dhsStr(l,t,Tmax+1) * \mu^{s^h}_l \\ * \omega^s - dd(l,t,Tmax+1) * \mu^b_l * \omega^b \textbf{ else} \end{cases}$$

$$gainI(R_{l,l,t,t'}) = \begin{cases} -1 \textbf{ if } a^f_{l,t} = 0 \\ gainC(R_{l,l,t,t'}) + (Tmax - t + 1) * \mu^{s^f}_l * \omega^s \textbf{ else} \end{cases}$$

$$gainI(B_{l,l',t,t'}) = \begin{cases} -1 \textbf{ if } a^f_{l,t} = 0 \\ gainC(B_{l,l',t,t'}) + (Tmax - t + 1) * \mu^{s^f}_l * \omega^s \textbf{ else} \end{cases}$$

3.4 Efficient data structure

The evaluation of $gainC$ being computationally expensive, it is important to avoid unnecessary evaluations. Adding a sequence to a plan updates variables and thus could impact gains associated with unitary actions. However, it is not necessary to recompute the gain for each unitary action but only for the ones impacted by the sequence.

We use a matrix $MgainC$ that permits to directly obtain the value of $gainC$ for a given sequence. Of course, the initialisation of this matrix has a cost but is very efficient afterwards. Algorithm 3 describes how matrix $MgainC$ is updated when a sequence is applied to a plan. Note that to ease the understanding of the algorithm we do not detail the update of the *next* value but every time we update a $MgainC$ value we also update its corresponding *next* action. Once $MgainC$ values are up to date, it is necessary to update $MgainI$ which is the associated matrix to $gainI$ as $MgainC$ is associated to $gainC$. This update is simpler. It has to be updated for all actions that are initiated from a site whose location appears in the last applied sequence. It also has to be updated for all actions whose $MgainC$ have been updated.

Algorithm 3 Update of $MgainC$

Input: a plan P, a sequence seq of size n
$ToUp \leftarrow \emptyset$ ▷ A set to store locations
forall $X_{l,l',t,t'} \in seq$
  $ToUp \leftarrow ToUp \cup \{l\} \cup \{l'\}$
end forall
$m \leftarrow X_{*,*,*,*}$
while $m \neq \emptyset$ **do**
  $Y_{l,l',t,t'} \leftarrow argmax_{X_{l'',l''',t'',t'''} \in m} t''$ ▷ reverse chronological
  if $l' \in ToUp$ **then**
    if $gainC(Y_{l,l',t,t'}) \neq MgainC(Y_{l,l',t,t'})$ **then**
      Update $MgainC(Y_{l,l',t,t'})$
      $ToUp \leftarrow ToUp \cup \{l\}$
    end if
  end if
  Remove $Y_{l,l',t,t'}$ from m
end while

4 Experiments

This section presents experiments carried out to evaluate and compare BIS with the MIP model. Note that, due to space restrictions, we only report experiments on a subset of the instances that have been tested.

4.1 Instances

Experiments are carried out on a set of pseudo-random instances that share the same supply chain network topology. This topology follows the 3-level tree structure presented in Figure 1 and consists of a single distribution centre (*DC*) which also acts as a repair centre, 33 intermediate hubs, and 66 peripheral stores. Schedules and lead times for supply, return and repair routes are also identical for all instances and so is the demand frequency. Instances also share the same unit cost values: a demand not met in time (a backorder) costs 1000, a repair costs 100 and storing an item costs 1 in the *DC* and 10 elsewhere. All these data have been extracted from a real-life problem instance developed for managing field service operations in the telecommunications sector. Note that in this instance, transfer costs are not taken into consideration and we thus set the unit transfer cost to 0.

We assume that the initial plan of actions is empty. However, we introduce variability in our instances by varying the initial volume of healthy items and their distribution across the sites. We use two configuration schemes to this effect. First, the initial number of healthy items is set as a proportion of the number of demands forecast over the time horizon and each instance is generated using one of the following ratios: 100% (category `High`), 50% (category `Med`) or 0% (category `Low`). Second, the initial volume of healthy items is either completely allocated to the *DC* (category `DC`), either randomly allocated to the peripheral stores (category `Sites`), or evenly distributed between the *DC* and the stores (category `Mix`). We thus get seven classes of instances since the initial distribution of healthy items is irrelevant for category `Low` (no healthy item is available).

4.2 Results

All computational experiments are carried out on a computer with an Intel Core i5-3380M processor (2.90GHz and 8GB of RAM). BIS is programmed in C++ and compiled using g++ for this architecture while the MIP model is solved using *CPLEX 12.6*. The cutoff time limit is set to 1 hour for MIP and 100 iterations for BIS.

Table 2 presents the results obtained by running MIP and BIS over the 7 classes of instances for all possible 0/1 weightings of the fitness function. For each instance and each weighting, we ran BIS 10 times and obtained the same best fitness value (i.e., the standard deviation is null). Each row corresponds to one instance of a specific class-weighting pair. The first two columns identify each class of instances based on its initial inventory level (category `High`, `Med` or `Low`) and inventory distribution (category `DC`, `Mix` or `Sites`). The third column indicates the values used for the triplet of weights $\langle \omega^b, \omega^{m^r}, \omega^s \rangle$. We recall that a weight set to 0 cancels the corresponding cost component in the fitness function. Note that we omit transport weight $\omega^{m^{hf}}$ since the unit cost is set to 0 in the studied instances thus making it ineffective.

The remaining columns provide, respectively, the best bound returned by MIP (column *MIP - Fitness*), the run time of MIP in milliseconds (column *MIP - Time*), the best fitness score returned by BIS (column *BIS - Fitness*), and the average run time of BIS in milliseconds over the 10 runs (column *BIS - Time*). A dash symbol '−' indicates that MIP did not find a solution before the cutoff time limit. Values in bold indicate cases where the fitness score obtained with BIS is lower than the fitness score obtained with MIP.

Table 2. Comparison between MIP and BIS

		Weights	MIP		BIS	
			Fitness	Time	Fitness	Time
High	DC	<1,0,0>	0	57000	0	22002
		<0,1,0>	-	-	**0**	42343
		<0,0,1>	99990	20500	99990	12129
		<0,1,1>	101574	23100	101574	8029
		<1,1,0>	0	32480	0	22126
		<1,0,1>	112200	-	112200	28040
		<1,1,1>	113784	-	113784	25625
	Mix	<1,0,0>	0	26220	0	14726
		<0,1,0>	0	10780	0	3986
		<0,0,1>	133862	2980	133862	6979
		<0,1,1>	135446	2840	135446	4133
		<1,1,0>	0	29500	0	14340
		<1,0,1>	143562	-	143562	19140
		<1,1,1>	145146	-	145146	16473
	Sites	<1,0,0>	27000	2730	27000	6922
		<0,1,0>	0	2070	0	4032
		<0,0,1>	256128	2030	256128	6282
		<0,1,1>	257712	2030	257712	3952
		<1,1,0>	35600	3430	35600	7171
		<1,0,1>	285983	3510	285983	9482
		<1,1,1>	295909	3930	295909	7805
Med	DC	<1,0,0>	0	38000	0	22094
		<0,1,0>	-	-	**0**	3950
		<0,0,1>	80751	35410	80751	11040
		<0,1,1>	82335	31500	82335	8250
		<1,1,0>	36300	-	36300	22150
		<1,0,1>	92961	-	92961	28457
		<1,1,1>	129756	-	129756	25190
	Mix	<1,0,0>	0	25680	0	18567
		<0,1,0>	0	120500	0	3985
		<0,0,1>	88669	18250	88669	8497
		<0,1,1>	90253	14240	90253	5437
		<1,1,0>	36300	-	36300	18771
		<1,0,1>	100339	-	100339	23701
		<1,1,1>	-	-	**137134**	21512
	Sites	<1,0,0>	93000	11730	93000	14760
		<0,1,0>	0	4480	0	3984
		<0,0,1>	114648	6051	114648	6273
		<0,1,1>	116232	3900	116232	4338
		<1,1,0>	129300	-	129300	14068
		<1,0,1>	217263	19240	217263	17083
		<1,1,1>	254058	-	254058	16292
Low	-	<1,0,0>	11550000	-	11550000	9441
		<0,1,0>	0	2000	0	3993
		<0,0,1>	66198	8410	66198	6536
		<0,1,1>	67782	8140	67782	4012
		<1,1,0>	11603200	-	**11596200**	9649
		<1,0,1>	11623128	-	11623128	10388
		<1,1,1>	11669526	-	11669526	10233

4.3 Analysis

First, we point out that, in all cases, BIS always reaches the best fitness score found by MIP. In some cases, MIP cannot find the optimal solution and returns an upper

bound or no bound at all. In 4 of these cases, BIS returns a solution that improves this bound. Note that when the only activated weight is repair weight ω^{m^r}, the fitness value is equal to 0, as expected. As for run time, BIS is generally faster than MIP except for category `High` and `Sites`. Besides, MIP reaches its time limit in 18 cases (1 hour) whereas BIS always returns a very good solution in less than 1 minute. Overall, these results confirm the scalability of BIS and its ability to attain optimal or near-optimal solutions in a reasonable time.

5 Towards Distribution Policies

Different weightings of the fitness function give rise to different behaviours in the way items get distributed over time in the supply chain. Figure 2 illustrates the temporal evolution of various supply chain metrics for two different weightings.

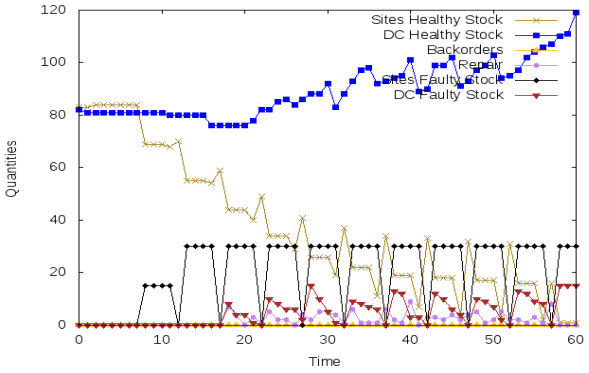
Weighting $\langle 1, 0, 1 \rangle$ aims at minimising both backordering and storage. As expected, all demands are met in time. We also notice that all faulty items are sent back to the *DC* and repaired to avoid storage costs. Healthy items are kept in the *DC* as long as possible since storing items in other sites is more expensive. Thus, this weighting can be seen as a solve demands just-in-time and repair everything management policy.

Weighting $\langle 1, 1, 0 \rangle$ aims at minimising both backordering and repair. As expected, all demands are met in time. The activation of repair objective implies that faulty parts are repaired only if needed to satisfy a demand later on. The deactivation of storage cost implies that items can be sent to sites at any time as it will not impact the fitness function. Thus, this weighting may be viewed as a solve demands and repair as less as possible management policy.

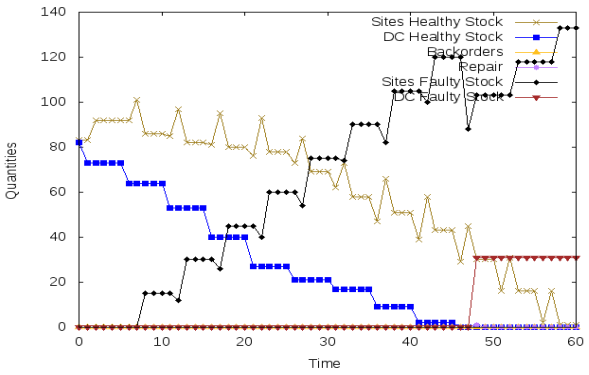
6 Conclusion

This article presented BIS, a metaheuristic based on sequences of unit actions to solve tactical distribution planning problems in closed-loop supply chains. We detailed the metaheuristic and the data structure used to limit the computational time. We also proposed an experimental analysis that shows the performances of BIS. Finally, we discussed the impact of the weights of the fitness function and the possibility of implementing different management policies.

As future work, we plan to investigate how to enforce distribution policies specified using indicators based on appropriate weightings of the fitness function. We also plan to investigate the robustness of our approach and to evaluate how it deals with uncertainties.



(a) `High`, `Mix`, $\langle 1, 0, 1 \rangle$, $Fitness = 33423$.



(b) `High`, `Mix`, $\langle 1, 1, 0 \rangle$, $Fitness = 100$.

Fig. 2. Impact of two different weightings